Canon

X-07

HAND HELD COMPUTER

PROGRAMMING FOR BEGINNERS

Canon

X-07

HAND-HELD COMPUTER

PROGRAMMING FOR BEGINNERS

CANON will not take responsibility for monetary damages, loss of profits, and any claims from a third party due to the use of the program samples in this manual.

This manual is a programming guide for people who want to freely use the X-07. On how to operate the keyboard and how to input a program, refer to the User's Guide.
Programming is an operation which puts a job you want the computer to perform into an executable form. For example, how to make the computer understand such instructions as "I will now read the receipts, so give me the total" and "Play back this musical selection", are explained with actual programs.

Table of contents

1. To create a program

1.1 What is a program ____ 3
1.2 Flowchart ____ 5
1.3 What is a good program ____ 9
1.4 How to save a program ____ 12

2. Basics of programming

2.1 The most basic instructions ____ 17
2.2 Reading data from the keyboard ____ 20
2.3 Changing the flow of control ____ 25
2.4 Making decisions ____ 28
2.5 Handling loops ____ 31
2.6 Reading data in a program ____ 33
2.7 Array variables ____ 37
A One-dimensional array
B Two-dimensional array
2.8 Handling character data ____ 46
A Connecting characters
B Reading character data
C Functions which handle character data
2.9 Defining functions ____ 54
2.10 Subroutines ____ 56

3. Practical programs

3.1 Turnover list ____ 63
3.2 Equalized repayment of principal and interest ____ 70
3.3 Scores of a golf competition ____ 74
3.4 Controlling records ____ 81
A How to create a record file
B Calculating statistics from a record
3.5 Telephone directory file ____ 91
A How to create a telephone directory file
B How to read the telephone directory file
C How to retreive data from the telephone directory file
3.6 Inventory control ____ 100

4. Programs which make the best use of the features of the X-07

4.1 Character display ——— 111
A Digital timer with an alarm
B Memory dump
C Mirror image
D Setting a character

4.2 Playing back a musical selection ——— 128
A Twinkle, twinkle little star
B Keyboard

4.3 Graphic display ——— 135
A Coordinate axes and dots
B Moving dot
C Drawing circles

4.4 Printer ——— 142
A Printing characters
B Graphically printing characters
C Drawing a pattern

4.5 Optical couple ——— 151
A Transmitting numeric values
B Transmitting characters
C Transmitting a file
D Conversation

5. Game program

5.1 Summary of the game ——— 165

5.2 Explanation of the game ——— 170
A Main routine
B Subroutines
C The game list

5.3 Excecution ——— 194

6. Troubleshooting

6.1 Endless loop ——— 202

6.2 Problems with displays ——— 204

6.3 Numeric values ——— 208

6.4 Other hard-to-recognize errors ——— 210

Advices

A program without a clear purpose ____ 4
Do as I do ____ 11
FOR I=? ____ 45
Practical programs ____ 90
Identity of a computer ____ 120
File ____ 158
How to clearly state ambiguities ____ 193

1 To create a program

2 Basics of programming

3 Practical programs

4 Programs which make the best use of the features of the X-07

5 Game program

6 Troubleshooting

Chapter 1
To create a program

This chapter is for people who are new to creating BASIC programs. It discusses basic concepts of program and points to keep in mind when creating a BASIC program.
If you prefer to go right into creating programs to theory, you may skip this chapter and start at Chapter 2. However, please come back later and read this chapter carefully.

1.1 What is a program

As discussed in the User's Guide, a program is a series of statements with line numbers. However, it cannot be just any series of statements. It must be structured to correctly execute a specific job. The process of creating a program is referred to as "programming".

Following is an example of a program.

```
10 I=INT(RND(0)*10)
20 INPUT "NUMBER";NUM
30 IF I=NUM THEN 100
40 IF I>NUM THEN PRINT "TOO SMALL"
50 IF I<NUM THEN PRINT "TOO LARGE"
60 GOTO 20
100 PRINT "YOU WIN"
110 END
```

This program is a game in which you try to figure out an integer, between 0 and 9, generated by the computer. Don't worry if you do not understand the meaning of each statement. You will come to understand them as you go through this manual. Here, we just want you to get the feel of a program.
Type and execute the above program. We hope you enjoy it, and experience the joy of programming.

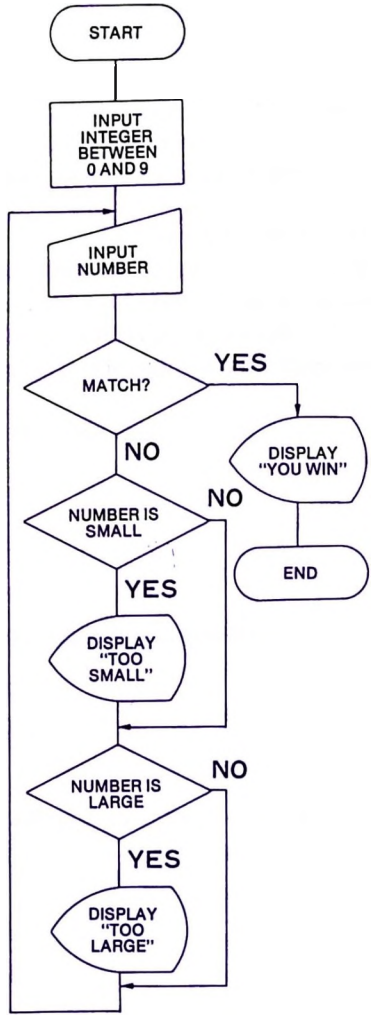
Note: If you do not know how to input this program or if an error is generated, please refer to "1. 4 BASIC programming" of the User's Guide.

A program without a clear purpose

When learning BASIC programming, the most difficult thing is not the actual grammar of the BASIC language, but to make clear the purpose of a program. For example, you know the grammar and vocabulary of your language, but if someone suddenly said to you, "Say something in your language", you would probably be lost for words. When you did come up with something to say, it would probably be something like, "It's a nice day today". If you were asked something with a clearer purpose such as "How do you say "hello" in your language?", you would have been able to speak more easily. This also applies when learning a foreign language such as BASIC. It is very difficult to learn and master the BASIC language when you do not have a clear aim of what you want to say in that language. The shortest way of learning to speak a language is by changing words of sample sentences. This also applies when trying to master the BASIC language. The shortest way is to change the various parts of programs someone has created.

1.2 Flowchart

It is difficult to grasp the general flow and the purpose of a program written in the BASIC language by just looking at the list of its contents. Longer the the program, more difficult it is to understand the program. For example, it is a very difficult task to understand a program of 500 or 1000 steps, just by looking at the list of its contents.
In such a case, it is easier to understand a program by drawing a diagram of its sequence of events. Such a diagram is called a "flowchart".
The diagram on the right is the flowchart of the program in the previous section. By writing such a flowchart you will see more clearly the purpose of a program, and how it will achieve this purpose.



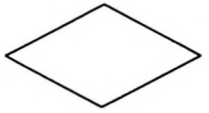
There are two purposes for writing a flowchart:

1. To clarify the flow of control before programming; and
2. To enable another person to understand your program, or just to keep as a record.

Please make it a habit to write a flowchart before writing a program. You probably think that you will be able to remember the logic of a program. Even if you have written it yourself, with a long program, you are bound to forget its logic in a course of time. In such a case, a flowchart is very handy.

There are number of symbols for writing a flowchart. There is no reason why you should use these symbols, but they are standard symbols. This section covers only the major symbols, and as you go through this manual, you will see how they are actually used.

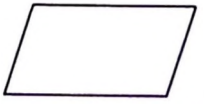
Process: Indicates various processing functions.



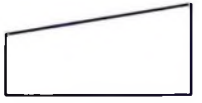
Decision: Used to branch the flowchart according to a set condition.

Preparation: Indicates various initial settings.

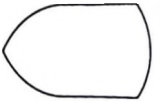
Defined process: Indicates a subroutine, etc.



Input/output: Indicates an output or an input function.



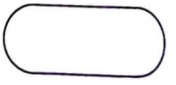
Manual input: Indicates manual input of data from the keyboard, etc.



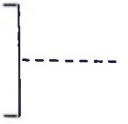
Display: Indicates displaying on the LCD.

Print: Indicates output to a printer, etc.

Linkage element: Indicates an outlet leading to or an inlet from another place in the flowchart.



Terminal: Indicates start, end, stop, delay, suspension, etc.



Remark: Used for inserting explanations.

1.3 What is a good program

A program can be written in many different ways to achieve one purpose. For example, the previous program can be modified, as shown below, and still achieve the same result.

```
10 I%=RND(0)*10
20 INPUT "NUMBER";NUM
30 IF I%=NUM THEN PRINT "YOU WIN":END
40 IF I%>NUM THEN PRINT "TOO SMALL"
50 IF I%<NUM THEN PRINT "TOO LARGE"
60 GOTO 20
```

You are wrong if you think one program is as good as another if the same result can be obtained. There is a big difference in excution speed and program size. There is such thing as a good program and a bad (?) program.
If you are going to create a program, let's create a good program. Well, what is a good program? Here's a list of what a good program should have.

1. **It accurately achieves the desired result.**
2. **It is easy to understand.**
3. **It is easy to correct.**
4. **It has high execution speed.**
5. **It is easy to use.**
6. **It is compact (not loose).**

To satisfy these requirements, keep the following in mind when writing a program.

1. The minimum condition of a program is to achieve the desired result. A program cannot be called a program if it does not do so.
2. It is important to write a program in an easy-to-understand style. Ease in understanding leads to ease in program correction (requirement 3). To make it easy to understand, insert appropriate

remarks, streamline the flow of control, etc.

3. To be able to easily correct a program, do not use too many multi-statement lines. Structure the program so that if you must make a correction, it will not affect other parts of the program.
4. To decrease execution time, write parts of the program which take time in machine language. Study the algorithm of loop processing to avoid wastes, set the often used variables at the beginning, and, use integer variables and multi-statements.
5. When creating a program, imagine how it will be executed. And ask someone to execute you program and ask for advice.
6. Use subroutines to streamline the program. However, try to avoid making the program too difficult to understand or too complex just for the sake of making it short.

To summarize these points:

1. **Have a clear purpose and clear idea of what you want to input and output.**
2. **Look at the whole flow of the program and not just one part of it.**
3. **Take your time when creating a program.**

Do as I do

Compared with the English, French, or German language, the BASIC language has much fewer grammatical rules and vocabulary. And, it seems as though, if you study whole heartedly, you could master it in 3 days. However, in the end, the problem that arises during actual programing is that, "Gestures mean nothing to the computer, and only the words input from the keyboard can lead to success or failure." As there is no such word as "implication", everything from A to Z must be specified correctly. A world without implications is very unnatural for human beings. But, a program consists of instructions to the computer, and if the instructions are ambiguous, the result can be quite unexpected. For example,

Instruction: "Do as I do."
Result: "Do as I do."
Instruction: "No, not yet."
Result: "No, not yet."
Instruction: "Now, stop repeating what I say."
Result: "Now, stop repeating what I say."

... it is like a comedy routine. There is no instruction such as "Do as I do" in the BASIC language. If there was one, a program written in the manner of the BASIC language for human being would be as follows:

Instruction: "After I say start, listen to my words. If the phrase "Now stop repeating what I say" is not included, repeat after me. If it is, stop repeating what I say, and listen to my next instruction. Start."

1.4 How to save a program

A program can be saved in the RAM or on a cassette tape.
If a program is short, or if you want to save it only temporarily, use the RAM. On the other hand, if you want to save it for a long time, use a cassette tape.
An area in memory (RAM) used for saving programs is called a "RAM file", and the FEST command is used to reserve its area in memory. Therefore, in order to save programs, the FSET command must be executed first.

Example:

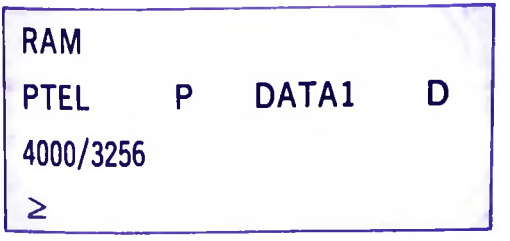
Type, **FSET 4000** (4000 bytes are reserved)

Then, **SAVE"file name"**

Now, a program with the specified file name can be saved in the RAM file. To call back the saved program, type

LOAD"file name"

Also, if you execute the DIR command, the contents of the RAM file can be displayed. For example:



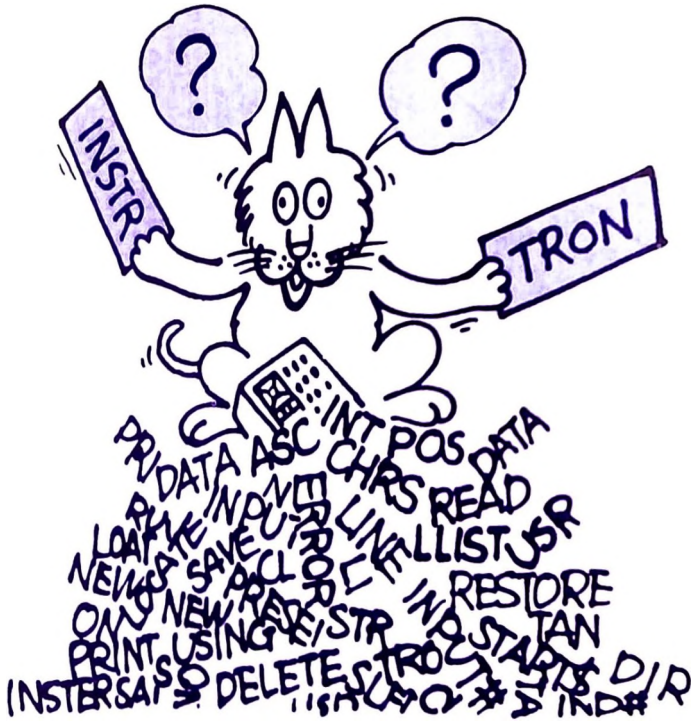
The display shows that the RAM file contains a program file called "PTEL" and a data file called "DATA", and 3256 bytes of unused

area is left in the RAM file. For details, please refer to "1.5 Memory of the X-07" of the User's Guide.
To save onto or to call a program from a cassette tape, the following commands are used:

CSAVE"file name"
CLOAD"file name"

Chapter 2
Basics of programming

This chapter gives several examples of program to help you learn about basics of programming and how to use basic instructions.



2.1 The most basic instructions

The following program calculates and displays the total sales for a given price and quantity.

```
10 REM TOTAL SALES
20 LET PRICE=50
30 LET NUM=8
40 LET SUM=PRICE*NUM
50 PRINT SUM
60 END
```

The explanation of this program is as follows:

10 REM TOTAL SALES

The REM statement is used to insert remarks in order to identify or clarify a program. It is not executed. Anything can be written after the instruction word "REM". Use REM statements to help you, or someone else, understand a program or parts of a program. Also, an apostrophe (') can be used instead of the instruction word "REM".

```
10  'TOTAL SALES
20  LET PRICE = 50
30  LET NUM = 8
40  LET SUM = PRICE*NUM
```

These are called "assignment statements". In other words, at Line 20, the value 50 is assigned to the variable PRICE, at Line 30, the

value 8 is assigned to the variable NUM and, at Line 40, the product of the contents of the variables PRICE and NUM is assigned to the variable SUM.
In an assignment statement, the instruction word "LET" can be omitted. In other words, the above statements can be written as:

```
20 PRICE = 50
30 NUM = 8
40 SUM = PRICE*NUM
```

It is easier to write as above, so the instruction word "LET" is often omitted.

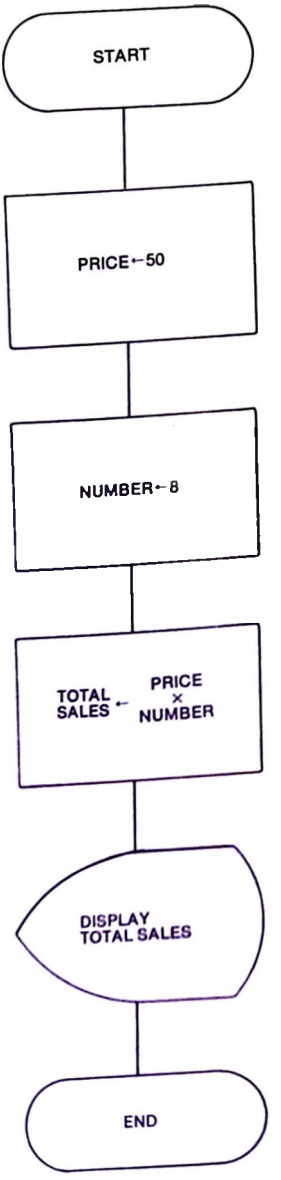
```
50 PRINT SUM
```

The print statement displays the contents of the variable SUM on the LCD. A question mark (?) can be used instead of the instruction word "PRINT". In other words, Line 50 can be written as:

```
50 ? SUM
```

```
60 END
```

The END statement terminates program execution. However, an END statement at the end of a program can be omitted. Therefore, in the above example, the program will be terminated even without Line 60.



Now, execute the program, and if the following is displayed, everything is OK.

```
RUN
 400
≥
```

New instructions

REM, '	Remark statement
LET	Assignment statement
PRINT, ?	Displays data on the LCD.
END	Terminates program execution.

2.2 Reading data from the keyboard

In the previous program, the price and the quantity were preset to 50 and 8, respectively. Now, let's modify the program so that any values you want can be calculated.

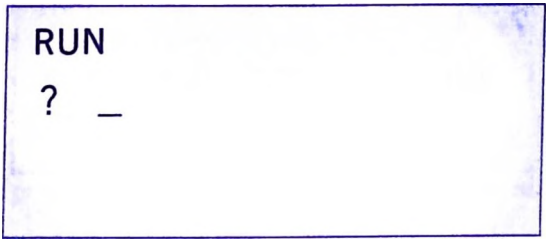
```
10 REM TOTAL SALES
20 INPUT PRICE
30 INPUT NUM
40 LET SUM=PRICE*NUM
50 PRINT SUM
60 END
```

The difference between this program and the previous program is that Lines 20 and 30 of the previous program have been changed to as follows:

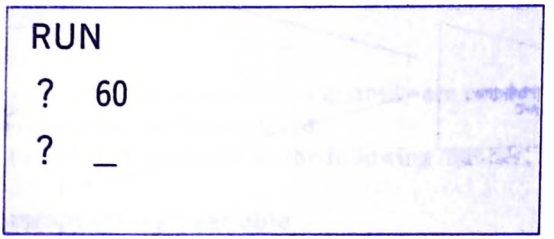
```
20 INPUT PRICE
30 INPUT NUM
```

The INPUT statement reads the data input from the keyboard and assigns them to the variables.

Now, execute the program.



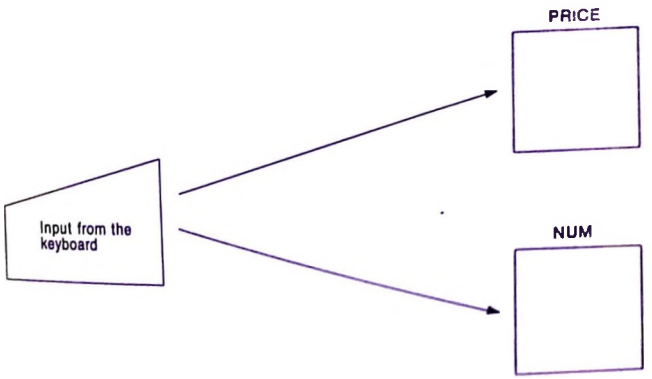
When the RUN command is input, a question mark (?) is displayed at the first position, and a flashing corsor is displayed at the third position. These two displays mean that the X-07 is asking you to input data from the keyboard. Please input 60.



The "?" and the cursor are displayed again. The first "?" was displayed by the execution of the INPUT statement of Line 20. The next "?" was displayed by the execution of the INPUT statement of Line 30. This time, please input 4.

```
? 60
? 4
  240
≥
```

In this example, 60 is assigned to the variable PRICE, and 4 to the variable NUM, and the product of the contents of these variables is assigned to the variable SUM.



In this program, only question marks are displayed, and a third party cannot tell what is going on. The result is only in number so that a third party cannot tell what the number means. Therefore, for the sake of kindness, let's modify the program to as follows:

```
10 REM TOTAL SALES
20 INPUT "PRICE";PRICE
30 INPUT "NUMBER";NUM
40 LET SUM=PRICE*NUM
50 PRINT "SUM";SUM
60 END
```

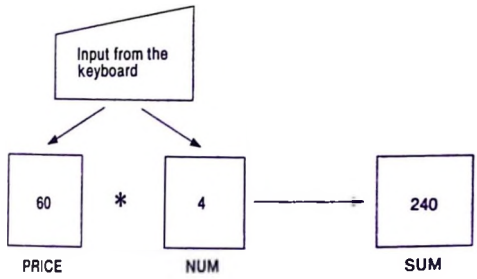
When this program is executed, the following is displayed:

```
PRICE ?  40
NUMBER ?  5
SUM  200
≥
```

Now, it is very clear that a price and a quantity are need to be input so that the total sales can be displayed.
By setting the INPUT statement in the following manner,

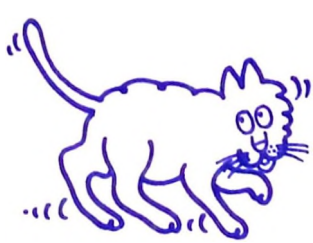
INPUT "prompt string"; variable

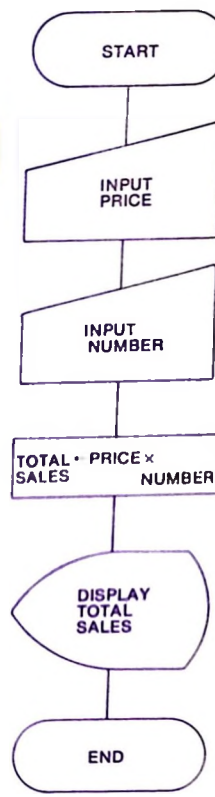
a prompt string (message indicating what to input) and "?" are displayed at the beginning of a line so that anyone can understand what values to input. The PRINT statement is not an input instruction, but it can also be set in the same manner.



New instruction

INPUT Inputs data from the keyboard.





2.3 Changing the flow of control

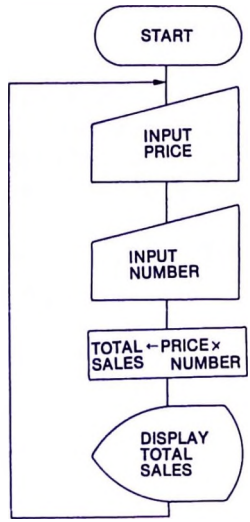
The last program terminated after calculating one total sales. Let's modify the program so that it will calculate several times.

```
10 REM TOTAL SALES
20 INPUT "PRICE";PRICE
30 INPUT "NUMBER";NUM
40 LET SUM=PRICE*NUM
50 PRINT "SUM";SUM
55 GOTO 20
60 END
```

In this program, the following statment was added:

55 GOTO 20

Normally, a program is executed in sequence from the lowest line number. The GOTO statement changes the flow of control, so that after Line 55 is executed, the next line to be executed is not Line 60, but Line 20.



Let's execute this program.

```
PRICE?  25
NUMBER?  5
SUM  125
PRICE?  23
NUMBER?  7
SUM  161
PRICE?  70
NUMBER  4
SUM  280
PRICE?  70
```

As you can see, this program is very convenient: it calculates repeatedly. However, there is just one little problem: Line 60 is never executed, so the program is never terminated. It is, therefore, necessary to forcibly bring it to its end by depressing the ON/BREAK key. Please do so.

```
Break in 30
```

The above message is displayed, and program execution is stopped. The message indicates that the program execution has been interrupted at Line 30 (the line number may vary). If you want to resume program execution, type

```
CONT
```

New instructions

GOTO n	Transfers control to Line n.
CONT	Resumes program execution from where it has been interrupted.

2.4 Making decisions

Depressing the ON/BREAK key is a clumsy way to terminate program execution. It is more natural to terminate program execution with an END statement. Let's correct the program so that it will terminate after calculating 3 times.

```
10 REM TOTAL SALES
15 I=0
20 INPUT "PRICE";PRICE
30 INPUT "NUMBER";NUM
40 LET SUM=PRICE*NUM
50 PRINT "SUM";SUM
52 I=I+1
55 IF I<3 THEN GOTO 20
60 END
```

Here, the variable I is used as a counter. First, at Line 15, the counter is set to 0. However, as 0 is initially assigned to all the variables, this line can be omitted. At Line 52, after each time the total sales is calculated, 1 is added to the counter. In other words, 1 is added to the contents of the variable I, and the sum is assigned to the variable I. Every time Line 52 is executed, the value of the variable changes as follows:

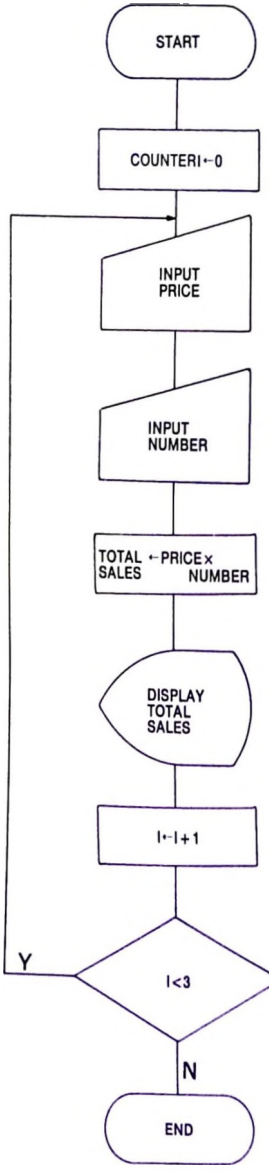
0→1 First time
1→2 Second time
2→3 Third time

.
.

Line 55 means,"If the value of I is less than 3, go back to Line 20. If not, execute the next line." An IF statement makes decisions based on a given condition, and changes the flow of control according to whether the condition is met or not. Let's execute this program.

```
First time    PRICE?   50
             NUMBER?    4
             SUM    200
Second time   PRICE?   60
             NUMBER?    3
             SUM    180
Third time    PRICE?   75
             NUMBER?    6
             SUM    450
              ≥
```

As you can see, the program was terminated after 3 executions.



New instruction

IF ~ THEN Makes decisions.

2.5 Handling loops

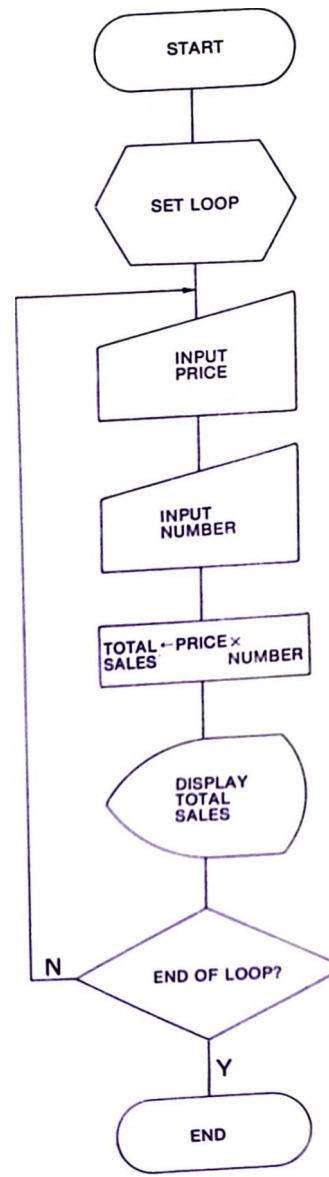
A sequence of instructions which is executed repeatedly a number of times, as in the previous program, is called a "loop". There is a much simpler way of creating a loop: by using the FOR ~ NEXT statements. Change Lines 15 and 55, and take out Line 52.

```
10 REM TOTAL SALES
15 FOR I=1 TO 3
20 INPUT "PRICE";PRICE
30 INPUT "NUMBER";NUM
40 LET SUM=PRICE*NUM
50 PRINT "SUM";SUM
55 NEXT I
60 END
```

The statements between the FOR and the NEXT statements are executed 3 times. To put it more correctly, the value of I is initially set to 1, and every time the statements between the FOR and the NEXT statement are executed, it is incremented by 1. When the value of the variable I is greater than 3, program execution goes out of the loop, and executes Line 60.
Please execute the program. Just as before, after total sales is calculated 3 times, the program is terminated.

New instructions

FOR ~ NEXT	Executes a loop.



2.6 Reading data in a program

The following program reads student marks and calculates the total.

```
10 REM MARKS
20 DATA 58,60,90,60
30 READ X1,X2,X3,X4
40 S=X1+X2+X3+X4
50 PRINT S
60 END
```

In this program, the marks are read from within the program and not from the keyboard. In other words, numeric values specified by the DATA statement are read into the variables specified by the READ statement. In this grogram, values 58, 60, 90 and 60 are read into the variables X1, X2, X3 and X4, respectively.

```
DATA 58, 60, 90, 60
       ↓   ↓   ↓   ↓
READ X1, X2, X3,, X4
```

In contrast to the INPUT statement which reads data from the keyboard, the READ statement reads data from the DATA statement. Let's execute this program.

```
RUN
 268
≥
```

Everything is OK if the above is displayed.
This program can be rewritten as follows:

```
10 REM MARKS
20 DATA 58,60,90,60
25 FOR I=1 TO 4
30 READ X
40 S=S+X
45 NEXT I
50 PRINT S
60 END
```

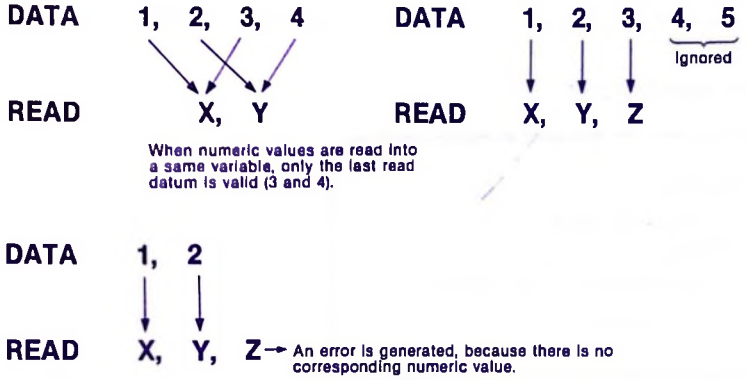
In this program, the variable X reads numeric values from the DATA statement in sequence.

```
20  DATA 58, 60, 90, 60
30  READ X
```

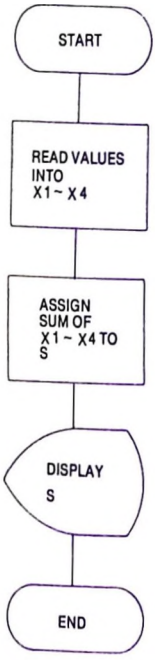
At Line 40, the read numeric values are added and assigned to the variable S. The values of S and X change as follows:

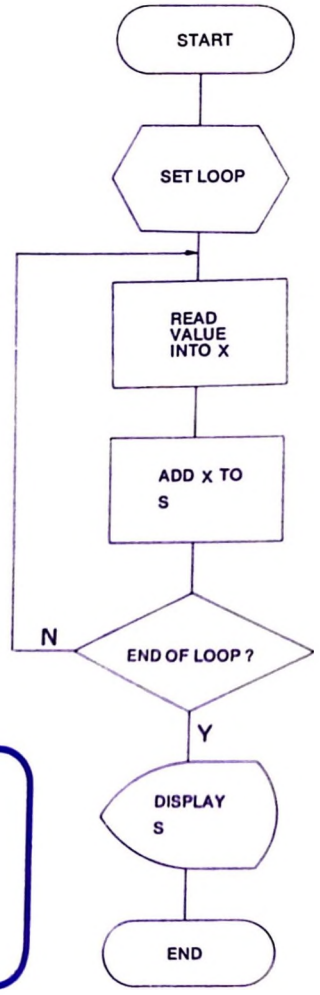
	S	X	S
I = 1	0 + 58 →		58
I = 2	58 + 60 →		118
I = 3	118 + 90 →		208
I = 4	208 + 60 →		268

In this method, only one variable is need to handle as many numeric values as you wish. It is very convenient.
Here are some examples of corresponding READ and DATA statements.


DATA 1, 2, 3, 4
READ X, Y
When numeric values are read into a same variable, only the last read datum is valid (3 and 4).
DATA 1, 2, 3, 4, 5
Ignored
READ X, Y, Z
DATA 1, 2
READ X, Y, Z
An error is generated, because there is no corresponding numeric value.

A DATA statement be placed anywhere in a program. Also, you can have more than one DATA statement. In such a case, data are read in sequence starting with the DATA statement with the lowest line number.





New instructions

DATA
Specifies data.

READ
Reads data from a DATA statement.

2.7 Array variables

A. One-dimensional array

In the following program, a student's number is input from the keyboard, and the mark of that student is displayed.

```
10 REM MARKS
20 DATA 60,90,80,75
30 READ X1,X2,X3,X4
40 INPUT "STUDENT NO.";N
50 IF N=1 THEN PRINT X1
60 IF N=2 THEN PRINT X2
70 IF N=3 THEN PRINT X3
80 IF N=4 THEN PRINT X4
90 END
```

The students' marks are specified in the DATA statement in order of their student numbers. Let's output the mark of the student number 2.

The following is displayed when this program is executed.

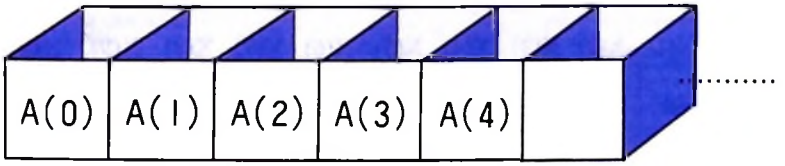
```
STUDENT NO. ?  2
 90
≥
```

You may have noticed that this program needs as many variables as there are students. It is quite adequate as long as the number of students are small, such as 4 in this program. However, it is too much if there are 100 students, for 100 variables would be needed. In such a case, an array variable is very convenient.
In a regular variable, only one datum can be assigned to a variable. However, many data can be assigned to an array variable. A variable is like a house with one occupant, but an array variable is like an apartment building with many occupants. An apartment is identified by its apartment number. The same thing applies for an array variable. An array variable has a number to identify its memory location. For example, the array variable A is given the following numbers:

A(0), A(1), A(2), A(3), ...

The numbers in the parentheses are called "subscripts".

Array variable A



Now, let's rewrite the previous program using an array variable.

```
10 REM MARKS
20 DATA 60,90,80,75,65,88,23,76,43,66
30 DIM X(10)
40 FOR I=1 TO 10
50 READ X(I)
60 NEXT I
70 INPUT "STUDENT NO.";N
80 PRINT X(N)
90 END
```

At Line 30, the array variable X is declared. An array variable must always be declared using a DIM statement before it can be used. In this case, an area from X(0) to X(10) is reserved. However, when a subscript is less than 10, an area is automatically reserved even when the DIM statement is omitted. Therefore, in this program, Line 30 can be omitted, if you wish.
At Line 50, data are read into the array variable X. If you also set the subscript to a variable, as it is in this case, an array variable can be

handled more easily. Data are read into the array variable X as follows:

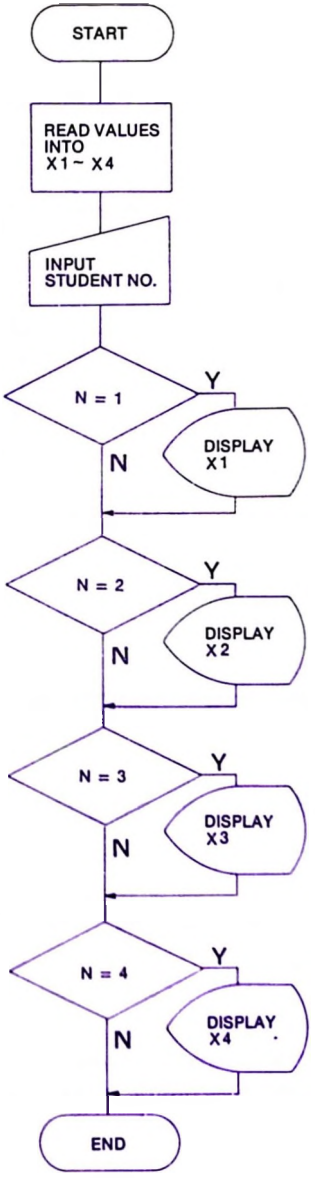
X(0)	X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)	X(9)	X(10)
0	60	90	89	75	65	88	23	76	43	66

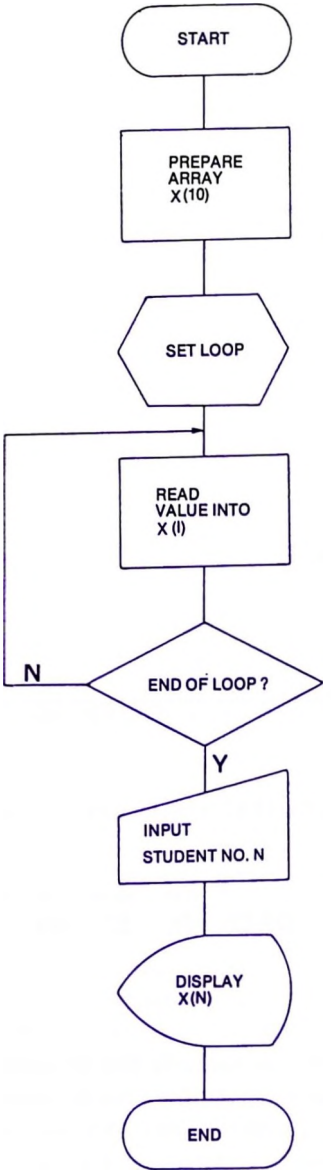
As X(0) is not used, the initial value of 0 is input. Let's execute the program.

```
RUN
STUDENT NO. ?  6
 88
≥
```

New instruction

DIM	Declares an array.


START
READ VALUES INTO X1 ~ X4
INPUT STUDENT NO.
N = 1
Y
N
DISPLAY X1
N = 2
Y
N
DISPLAY X2
N = 3
Y
N
DISPLAY X3
N = 4
Y
N
DISPLAY X4
END

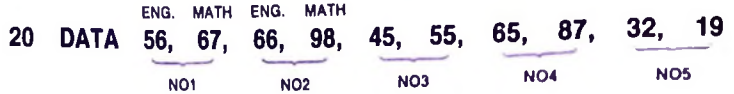

START
PREPARE ARRAY X(10)
SET LOOP
READ VALUE INTO X (I)
N
END OF LOOP ?
Y
INPUT STUDENT NO. N
DISPLAY X(N)
END

B. Two-dimensional array

In the next program, student numbers and subjects are input from the keyboard, and students' marks are displayed.

```
10 REM MARKS
20 DATA 56,67,66,98,45,55,65,87,32,19
30 DIM X(5,1)
40 FOR I=1 TO 5
50 FOR J=0 TO 1
60 READ X(I,J)
70 NEXT J
80 NEXT I
90 INPUT "STUDENT NO.";N
100 INPUT "ENG.(0) OR MATH.(1)";M
110 PRINT X(N,M)
120 END
```

At the DATA statement of Line 20, marks are lined up as follows:



The array variable X is declared by the DIM statment of Line 30. This time, there are two subscripts, separated by a comma. Such an array is called a "two-dimensional array". The memory locations of this two-dimensional array are reserved as folows:

X(5,1)

X(0,0)	X(1,0)	X(2,0)	X(3,0)	X(4,0)	X(5,0)
X(0,1)	X(1,1)	X(2,1)	X(3,1)	X(4,1)	X(5,1)

At Lines 40 to 80, data are read into the array variable from the DATA statement. Here, please note that one FOR ~ NEXT loop is inside another FOR ~ NEXT loop. Putting a loop within a loop is referred to as "nesting".

Nesting

```
FOR  I
  FOR  J
  NEXT  J
NEXT  I
```

It is illegal to structure a nesting in the following manner:

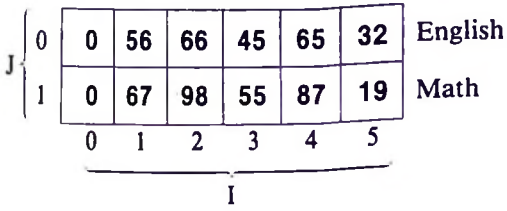
```
  FOR  I
FOR  J
  NEXT  I
NEXT  J
```

In nested FOR ~ NEXT loops, the contents of the inner loop variable changes quicker than the contents of the outer loop variable. In the present program, the contents of the variables I and J change as follows:

I	1	1	2	2	3	3	4	4	5	5
J	0	1	0	1	0	1	0	1	0	1

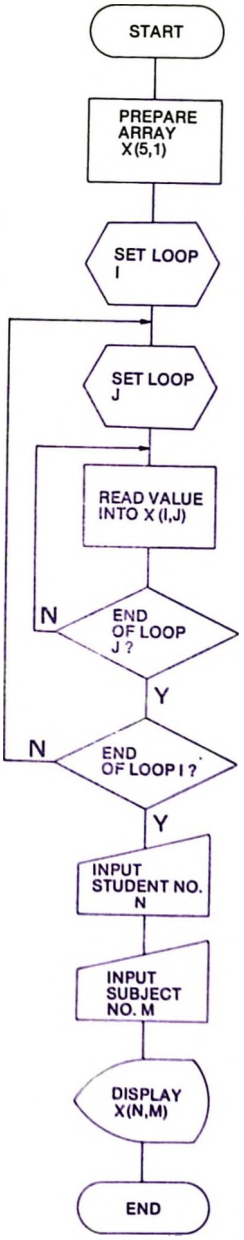
X(1,0) X(1,1) X(2,0) X(2,1)..... X(5,1)

When the program is executed, the following data are stored in the array variable X.



When the program is executed, the following is displayed.

```
STUDENT NO. ?  3
ENG. (0)  OR MATH. (1) ?  1
 55
≥
```



FOR I = ?

The variable used by the FOR statement is called a "counter", but why is "I" used so often? This is because before the BASIC language was created, there already was a language called FORTRAN with wide influence. In the FORTRAN language, I, J, K, L, M, and N are used for the integer-type variables, and the people who are used to using such letters continued out of habit to use I as the counter when using the BASIC language. In the BASIC language a variable name can be anything. A beginner might mistake "I" for "1", so A, or N might be more convenient, but it seems best to keep alive the accumulation of the past usage. This is because, most often, when you look at programs which have been created up to now, you can tell easily that I is the counter, and X, Y are coordinates of a screen, etc. As you use the BASIC language, you will make fewer mistakes such as "FOR 1 = 0 TO 3", and "NEXT 1"

2.8 Handling character data

A. Connecting characters

In the following program, a first name and a last name are input from the keyboard, and are displayed, connected together, on the LCD.

```
10 REM
20 INPUT "FIRST";F$
30 INPUT "LAST";L$
40 NAME$=F$+" "+L$
50 PRINT NAME$
60 END
```

As the data to be input from the keyboard are characters and not numeric values, the variables in the INPUT statements must be string variables. A dollar sign ($) is placed at the end of a string variable name.
At Line 40, the input names are connected with a blank space between them. As you can see, to connect character strings, the same symbol (+) as adding numbers are used.

Please execute the program.

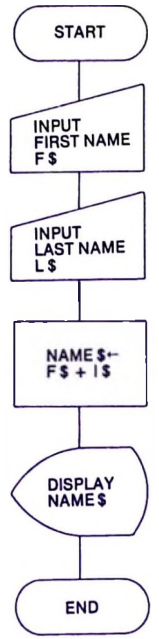
```
FIRST? MARY
LAST? SMITH
MARY SMITH
≥
```

In the above example, the character strings were stored in the variables as shown below.

F$ MARY + " " + L$ SMITH
→NAME$ MARY SMITH

New instruction

+	Connects character strings



B. Reading character data

Let's write a program in which a student's mark is displayed when his/her name is input from the keyboard.

```
10 REM MARKS
20 DATA RANDY,TANIA,TOM
30 DATA 50,90,20
40 DIM A$(2),A(2)
50 FOR I=0 TO 2
60 READ A$(I)
70 NEXT I
80 FOR I=0 TO 2
90 READ A(I)
100 NEXT I
110 INPUT "NAME";N$
120 FOR I=0 TO 2
130 IF A$(I)=N$ THEN 160
140 NEXT I
150 GOTO 110
160 PRINT A$(I);" ";A(I)
170 END
```

Reads names

Reads marks

At Line 60, data are read into the array variable A$ from the DATA statement. Please note that, here, string constants are specified in the DATA statement. Accordingly, the array variable must be a string variable.

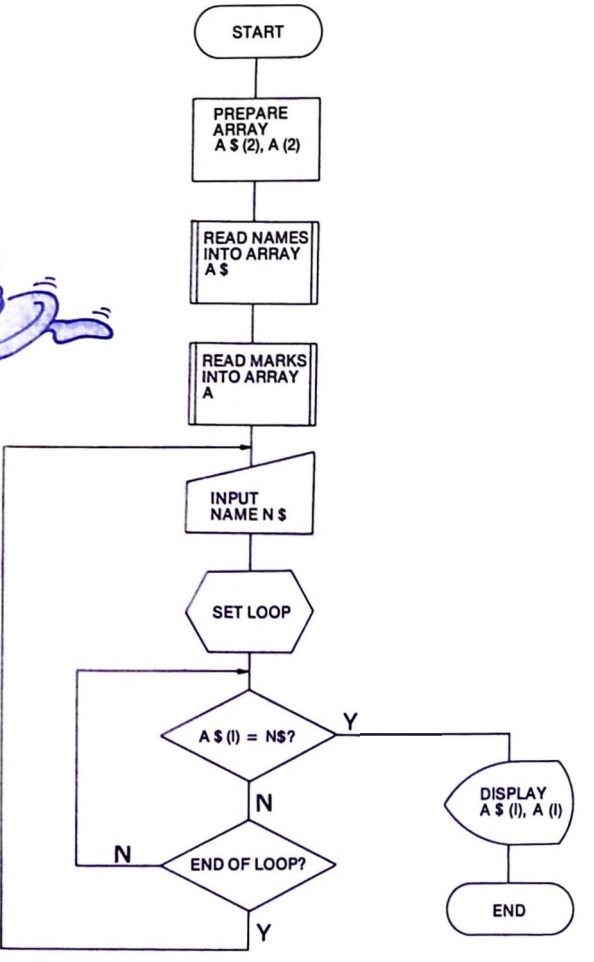
At Line 130, character strings are compared. If they match, the name and the mark are displayed at Line 160.

Following occurs when this program is executed.

```
NAME? TANIA

TANIA 90

≥
```



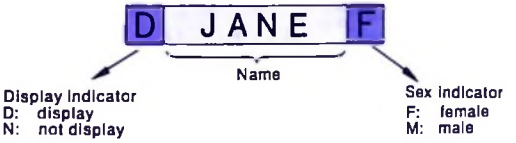
C. Functions which handle character data

Please study the following program.

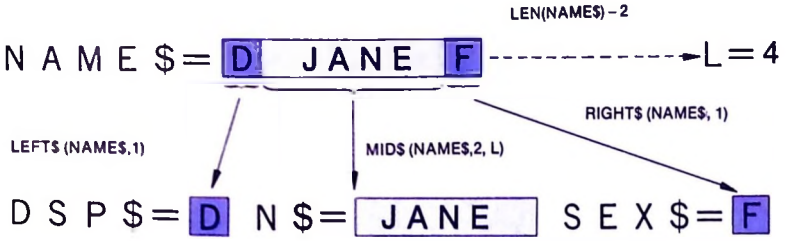
```
10 REM
20 DATA DJANEF,NTERRYM,DTOMM,DHELENF,NJI
MM
30 FOR I=1 TO 5
40 READ NAME$
50 DSP$=LEFT$(NAME$,1)
60 IF DSP$="N" THEN 120
70 SEX$=RIGHT$(NAME$,1)
80 L=LEN(NAME$)-2
90 N$=MID$(NAME$,2,L)
100 IF SEX$="M" THEN S$=" " ELSE S$="*"
110 PRINT S$+N$
120 NEXT I
130 END
```

In this program, data are read from the DATA statement, names are taken out, and then displayed.

Names are specified in the following form in the DATA statement.



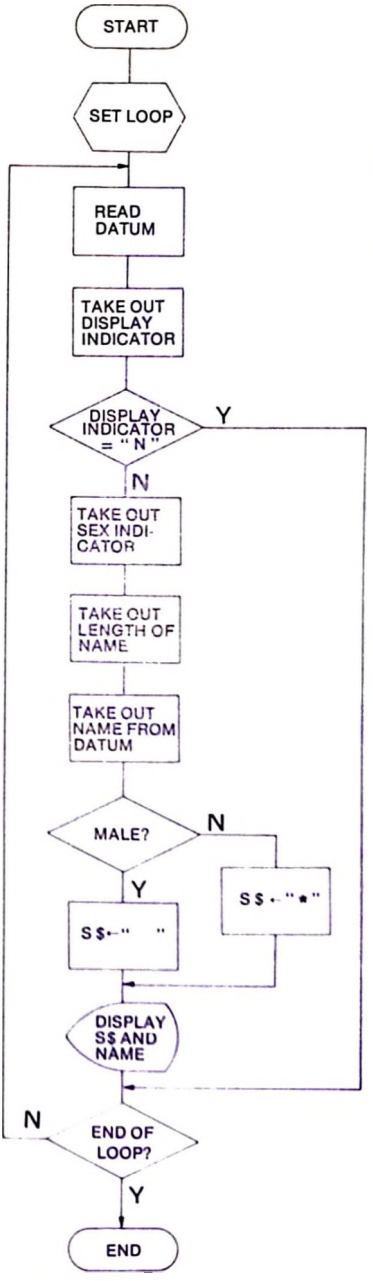
When the leading character is "D", the name is displayed. The last character denotes the sex. When it is a male name, a blank space is placed at the beginning of the displayed name, and an asterisk (*) is placed when it is a female name.
In this program, several functions which handle character data are used. The LEFT$, the RIGHT$, and the MID$ functions take out the left most, the right most and middle characters from a character string, respectively. The LEN function returns the number of characters in a character string.
When the program is executed, characters are stored in the variables as follows:



In other words, at Line 50, the left most character is taken out of the contents of the variable NAME$. At Line 70, the right most character is taken out of the contents of the variable NAME$. At Line 80, the length of the contents of the variable NAME$ less 2 (i.e. the length of a name) is calculated. At Line 90, characters are taken out of the variable NAME$ starting at the second character.
Please note that in the IF statement of Line 100, ELSE is used. The ELSE clause specifies the instruction to be executed when the specified condition cannot be satisfied.

The following is displayed when this program is executed.

```
* JANE
  TOM
* HELEN
≥
```



New instructions

LEN
Returns the length of a character string.

LEFT$
Takes out the left most characters of a character string.

RIGHT$
Takes out the right most characters of a character string.

MID$
Takes out a part of a character string.

2.9 Defining functions

The following program calculates the area and the circumference of a circle.

```
10 REM CIRCLE CALCULATION
20 DEFFN M(R)=3.14159*R*R
30 DEFFN N(R)=3.14159*2*R
40 INPUT I
50 X=FN M(I)
60 Y=FN N(I)
70 PRINT X,Y
80 END
```

Lines 20 and 30 define the functions M(R) and N(R) which calculate the area and the circumference of a circle from its radius R, respectively. At Line 40, the radius of a circle is input from the keyboard, and at Lines 50 and 60, the area and the circumference of the circle, respectively, are calculated using the defined functions.
By using the DEFFN function in this manner, you can freely define a function, and once a function is defined in a program, it can be used as many times as you wish within the program. For example, use the DEFFN function for often used expressions with very long format.

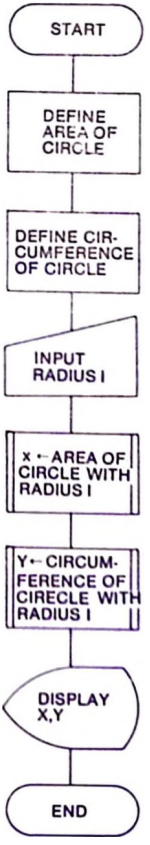
When this program is executed, the following is displayed.

```
? 7
  153.93791
  43.98226
>
```

Radius of circle

Area of circle

Circumference of circle



New instruction

DEFFN Defines a function.

2.10 Subroutines

The following program returns the maximum and the minimum values of a set of numbers.

```
10 REM MAX MIN
20 INPUT "NO. TIMES INPUT";N
30 DIM X(N)
40 FOR J=1 TO N
50 INPUT "NUMBER";X(J)
60 NEXT J
70 GOSUB 100
80 GOSUB 200
90 END
100 REM MAX
110 MAX=0
120 FOR K=1 TO N
130 IF MAX<X(K) THEN MAX=X(K)
140 NEXT K
150 PRINT "MAX VALUE";MAX
160 RETURN
200 REM MIN
210 MIN=100
220 FOR K=1 TO N
230 IF MIN>X(K) THEN MIN=X(K)
240 NEXT K
250 PRINT "MIN VALUE";MIN
260 RETURN
```

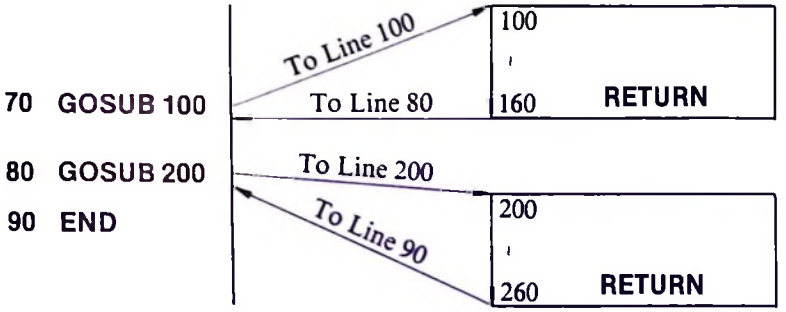
Subroutine which seeks the maximum value

Subroutine which seeks the minimum value

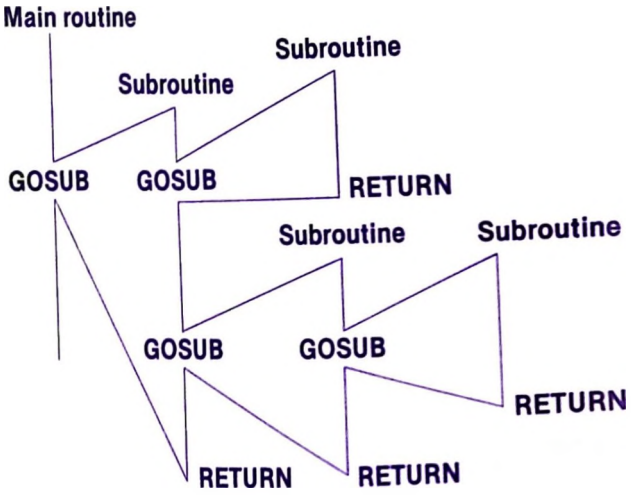
The maximum and the minimum values are displayed when numbers between 0 and 100 are input from the keyboard.

```
70  GOSUB 100
80  GOSUB 200
```

The GOSUB statement, like the GOTO statement, transfers control to the specified line number. However, when the RETURN statement is executed, control is returned to the statement following the GOSUB statement. A section of a program called in such a manner using a GOSUB statement is called a "subroutine".
In this program, the maximum and the minimum values are calculated at the subroutines called by the GOSUB statements in Line 70 and 80.



A subroutine can be called any number of times. Also, one subroutine can call another subroutine. An example of nested subroutines is shown below.



Nesting of subroutines

Subroutines are used to clearly divide functions, and to bring together similar sections.
Subroutines have the following merits. And, longer a program, greater their merits.

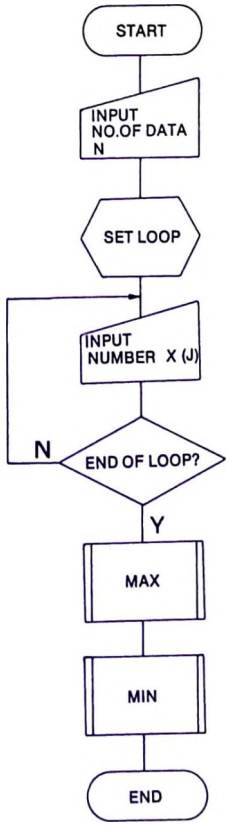
- **Functions can be divided clearly.**
- **Easy to debug.**
- **Brings together similar sections, so the program is shorter.**
- **A program can be written by several people.**

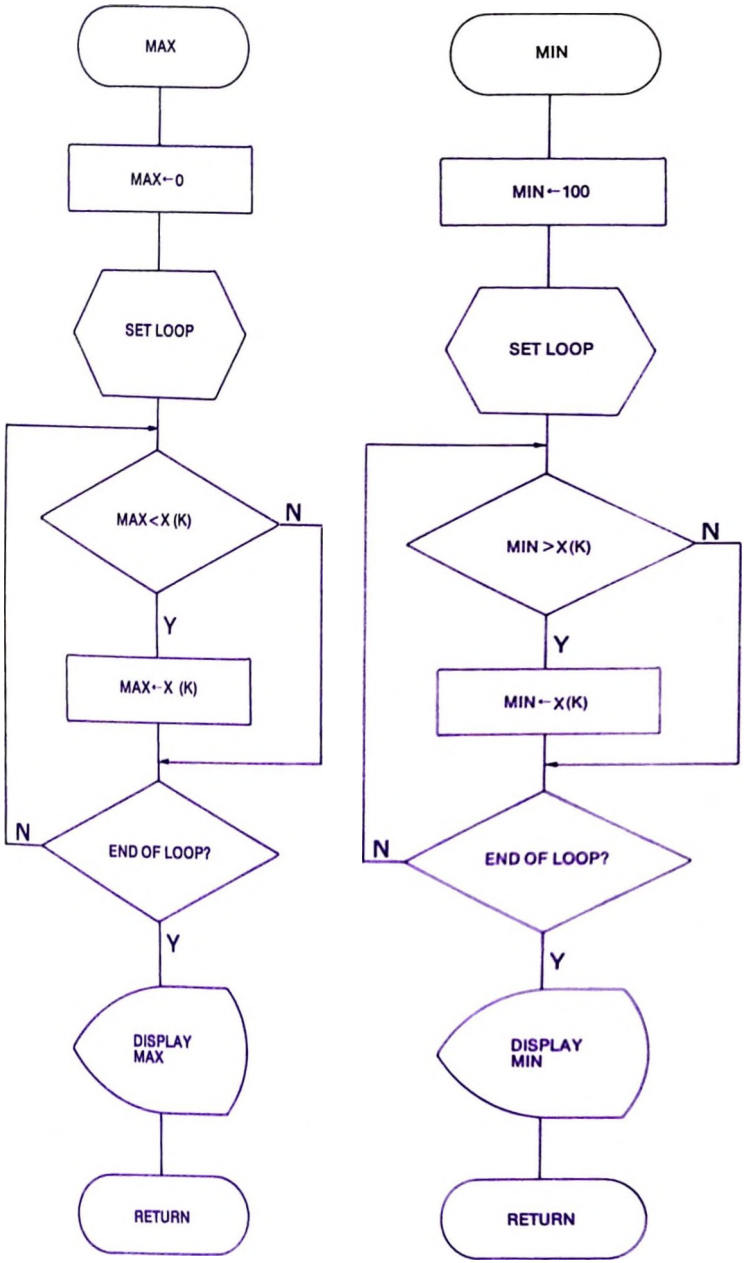
When this program is executed, the following is displayed.

```
NO. TIMES INPUT?  5
NUMBER?  34
NUMBER?  76
NUMBER?  18
NUMBER?  90
NUMBER?  73
MAX VALUE  90
MIN VALUE  18
≥
```

New instructions

GOSUB	Calls a subroutine.
RETURN	Terminates a subroutine.




MAX
MAX←0
SET LOOP
MAX<X(K)
N
Y
MAX←X(K)
N
END OF LOOP?
Y
DISPLAY MAX
RETURN
MIN
MIN←100
SET LOOP
MIN>X(K)
N
Y
MIN←X(K)
N
END OF LOOP?
Y
DISPLAY MIN
RETURN

Chapter 3
Practical programs

This chapter introduces several programs useful for commercial or home applications.

3.1 Turnover list

The following program calculates turnovers, and lists them in their decreasing order.

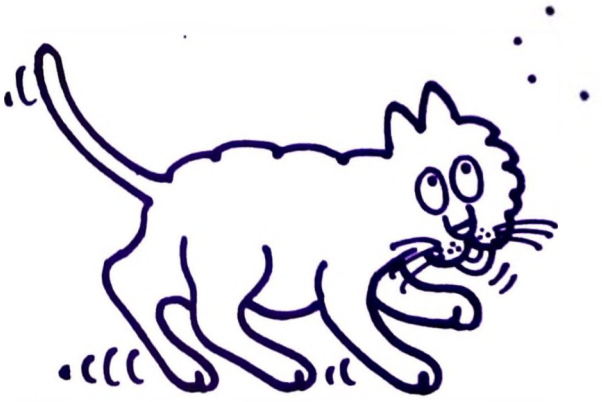
```
10 DIM B$(50),C(50),D(50),E(50)
20 B$=""
30 INPUT "PRODUCT";B$
40 IF B$="" THEN 110
50 I=I+1
60 B$(I)=B$
70 INPUT "PRICE";C(I)
80 INPUT "QUANTITY";D(I)
90 E(I)=C(I)*D(I)
100 GOTO 20
110 N=I
120 SW=0
130 FOR I=1 TO N-1
140 IF E(I)>=E(I+1) THEN 200
150 SW=1
160 W$=B$(I):B$(I)=B$(I+1):B$(I+1)=W$
170 W=C(I):C(I)=C(I+1):C(I+1)=W
180 W=D(I):D(I)=D(I+1):D(I+1)=W
190 W=E(I):E(I)=E(I+1):E(I+1)=W
200 NEXT I
210 IF SW=1 THEN 110
```

Inputs data and calculates turnovers

Sorts

```
220 FOR I=1 TO N
230 PRINT
240 PRINT "RANK";I
250 PRINT "PRODUCT";B$(I)
260 PRINT "PRICE";
270 PRINT USING "$#.##";C(I)
280 PRINT "QUANTITY";D(I)
290 PRINT "TURNOVER";
300 PRINT USING "$###.##";E(I)
310 NEXT I
320 END
```

Displays turnovers in decreasing order



After product names, prices, and quantities are input from the keyboard, the turnovers are calculated, sorted in their decreasing order, and output.

At Lines 20 to 100, product names, prices and quantities are input from the keyboard. The input process is terminated when a null string (depressing just the RETURN key) is input as a product name. Also, the turnover for each product is calculated at Line 90.

At Lines 110 to 210, turnovers are sorted. At Line 140, adjacent turnovers are compared, and if they are not in their increasing order, they are interchanged. When turnovers are interchanged, their product names, prices and quantities are also interchanged. Turnovers are compared N-1 times, and when there is an interchange, the variable SW is set to 1. In such a case, there is a possibility that the turnovers are not in their increasing order, so they are compared again N-1 times. Sorting is over when the variable SW remains at 0.

At Lines 220 to 310, ranks, product names, prices, and quantities are displayed in their decreasing order. The format of displaying prices and turnovers are set at Lines 270 and 300, respectively.

```
270  PRINT USING "$#.##";C(I)
300  PRINT USING "$###.##";E(I)
```

The above statements specify that a $ symbol be affixed to the price, and the turnover, a period be place before the last two digits, and the maximum number of digits be 3 (price) and 5 (turnover). The following is displayed when this program is executed.

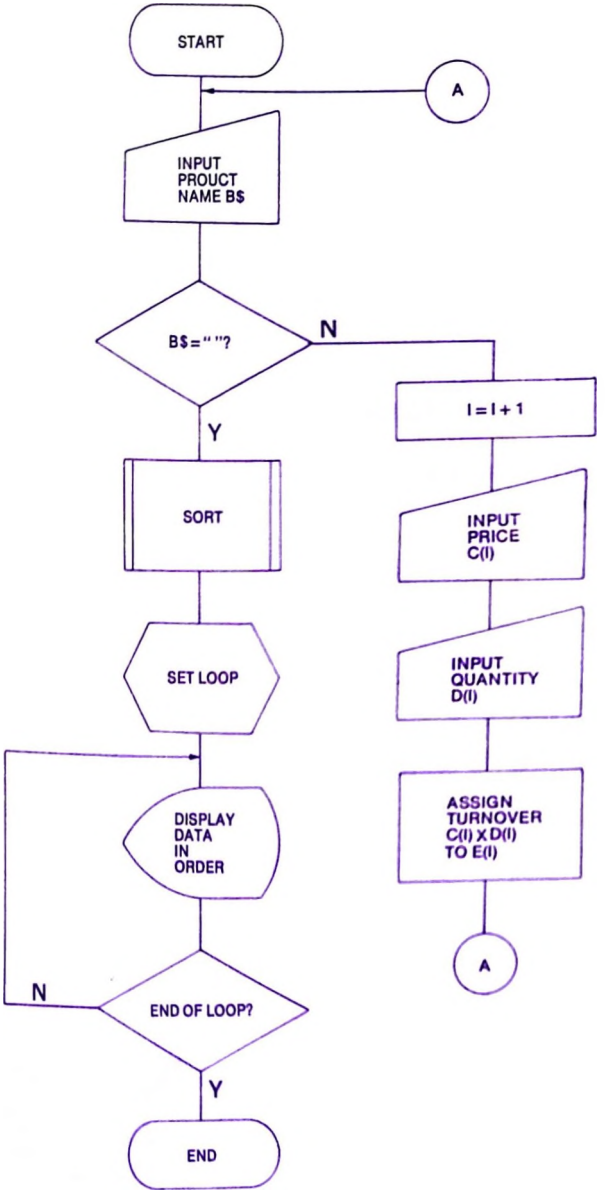
```
(Data input)
PRODUCT?  PEN
PRICE?  1.00
QUANTITIY?  150
PRODUCT?  NOTE
PRICE?  1.50
QUANTITY  90
PRODUCT?  FILE
PRICE?  3.00
QUANTITY?  55
PRODUCT?  PENCIL
PRICE?  1.20
QUANTITY?  100
PRODUCT?                ← Null (Only the RETURN key is depressed.)
```

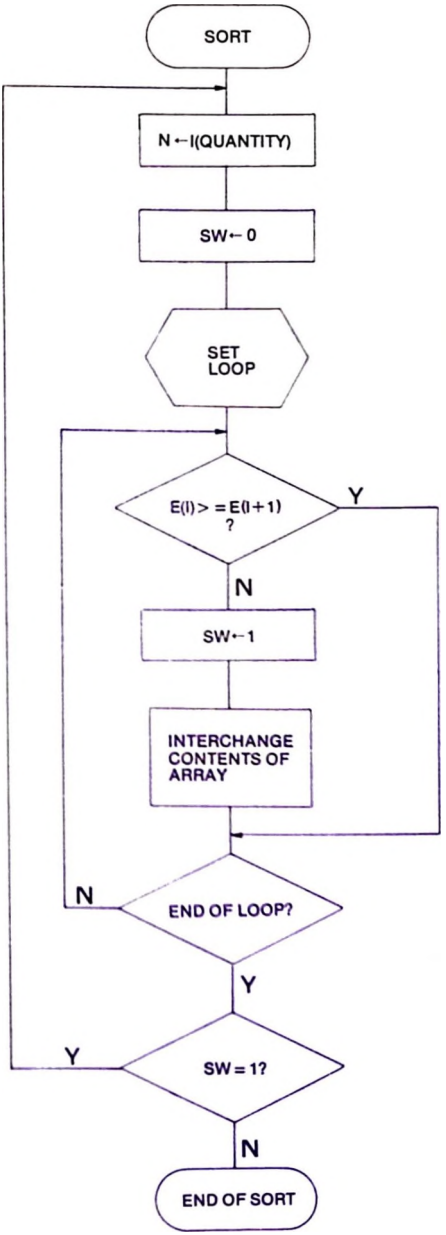
(Display of list in order)

RANK 1
PRODUCT: FILE
PRICE $3.00
QUANTITY 55
TURNOVER $165.00

RANK 2
PRODUCT: PEN
PRICE $1.00
QUANTITY 150
TURNOVER $ 150.00
RANK 3
PRODUCT: NOTE
PRICE $1.50
QUANTITY 90
TURNOVER $135.00

RANK 4
PRODUCT: PENCIL
PRICE $1.20
QUANTITY 100
TURNOVER $120.00


START
A
INPUT
PROUCT
NAME B$
B$=" "?
N
Y
I=I+1
SORT
INPUT
PRICE
C(I)
SET LOOP
INPUT
QUANTITY
D(I)
DISPLAY
DATA
IN
ORDER
ASSIGN
TURNOVER
C(I) X D(I)
TO E(I)
A
N
END OF LOOP?
Y
END



New instruction

PRINT USING
Specifies the display format.

3.2 Equalized repayment of principal and interest

Let's write a program for calculating the amount of equalized repayment of principal and interest, the amount of principal and the interest per repayment when the term of repayment, the interest rate per annum and the principal sum are input.
These amounts are calculated using the following expressions:

$$\text{Amount of equalized repayment of principal and interest} = \frac{\text{principal} \times \text{interest rate} \times (1 + \text{interest rate}) \times \text{years}}{(1 + \text{interest rate})^{\text{years}} - 1}$$

Interest per repayment = remainining amount of principal × interest rate per annum

Amount of principal = amount of equalized repayment of principal and interest to be paid back per repayment − interest at same repayment period

Although the calculation will introduce an error in terms of amount, the error will be adjusted by adding it to the first repayment.
The program is as follows:

```
10 REM REPAYMENT
20 DEFINT A,S
30 INPUT "TERM OF REPAYMENT";Y
40 INPUT "INTEREST PER ANNUM %";R
50 INPUT "PRINCIPAL";X
60 R1=R/100
70 A=X*R1/((1+R1)^Y-1)*(1+R1)^Y
80 PRINT
90 PRINT "AMOUNT OF EQUALIZED REPAYMENT"
;A;"DOLLARS"
100 Z=X
110 H=0
120 FOR I=1 TO Y
130 PRINT
140 S=Z*R1+.9
150 W=A-S
160 Z=Z-W
170 H=H+W
180 PRINT I;"REPAYMENT"
190 PRINT "PRINCIPAL";A-S;"DOLLARS"
200 PRINT "INTEREST";S;"DOLLARS"
210 NEXT I
220 P=X-H
230 PRINT
240 PRINT "REPAYMENT AMOUNT TO BE ADDED
1ST TIME";P;"DOLLARS"
250 END
```

Amount of equalized repayment

Amont of principal to be paid back

Remainder

Total amount of principal repayment

20 DEFINT A,S

This declares that the variables starting with A or S are integer-type variables. There are two ways of declaring the type of variable: using a type declaration statement or adding a declaration character.

	Type declaration statement	Type declaration character (example)
Integer type	DEFINT	% (A%)
Single-precision real number type	DEFSNG	! (B!)
Double-precision real number type	DEFDBL	#, No character (C3#, C)
String type	DEFSTR	$ (A$)

Please remember that there is also the INT function which truncates decimals.

Example: PRINT INT(3.59)
3

When the program is executed, the following is displayed.

TERM OF REPAYMNENT? 5
INTEREST PER ANNUM %? 9
PRINCIPAL? 100000
AMOUNT OF EQUALIZED REPAYMENT 25709 DOLLARS

Input

1 REPAYMENT
PRINCIPAL 16709 DOLLARS
INTEREST 9000 DOLLARS

2 REPAYMENT
PRINCIPAL 18212 DOLLARS
INTEREST 7497 DOLLARS

3 REPAYMENT
PRINCIPAL 19851 DOLLARS
INTEREST 5858 DOLLARS

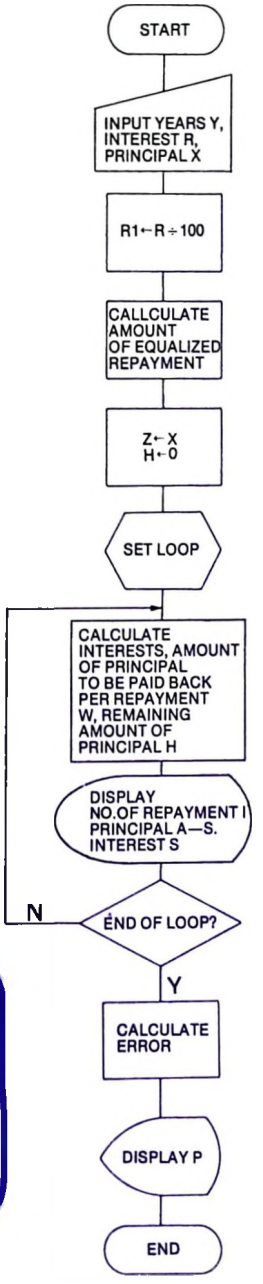
4 REPAYMENT
PRINCIPAL 21638 DOLLARS
INTEREST 4071 DOLLARS

5 REPAYMENT
PRINCIPAL 23585 DOLLARS
INTEREST 2124 DOLLARS

REPAYMENT AMOUNT TO BE
ADDED 1ST TIME 5 DOLLARS

New instructions

DEFINT	Declares integer-type variables.
DEFSNG	Declares single-precision real number-type variables.
DEFDBL	Declares double-precision real number-type variables.
DEFSTR	Declares string variables.



3.3 Scores of a golf competition

Let's write a program for calculating the gross and the net scres of each player by inputting the names, the handicaps, and the out and in scores of the players.
The gross and net scores are calculated as follows:

Gross score = out score + in score
Net score = gross score − handicap

Following is such a program.

```
10 REM GOLF
20 DIM N$(10),G(10,5)
30 N$=""
40 INPUT "NAME";N$
50 IF N$="" THEN 170
60 A=A+1
70 INPUT "OUT SCORE";N1
80 INPUT "IN SCORE";N2
90 INPUT "HCP";N3
100 N$(A)=N$
110 G(A,1)=N1
120 G(A,2)=N2
130 G(A,3)=N1+N2
140 G(A,4)=N3
150 G(A,5)=N1+N2-N3
160 GOTO 30
170 GOSUB 300
180 INPUT "CORRECTION NO.";X
190 IF X=0 THEN 270
200 INPUT "NAME";N$(X)
210 INPUT "OUT SCORE";G(X,1)
220 INPUT "IN SCORE";G(X,2)
230 INPUT "HCP";G(X,4)
240 G(X,3)=G(X,1)+G(X,2)
250 G(X,5)=G(X,3)-G(X,4)
260 GOTO 180
270 PRINT CHR$(12)
280 GOSUB 300
290 END
```

```
300 REM DISPLAY
310 FOR I=1 TO A
320 PRINT
330 PRINT I
340 PRINT "NAME";N$(I)
350 PRINT "OUT";G(I,1)
360 PRINT "IN ";G(I,2)
370 IF INKEY$="" THEN 370
380 PRINT "GROSS";G(I,3)
390 PRINT "HCP";G(I,4)
400 PRINT "NET";G(I,5)
410 IF INKEY$="" THEN 410
420 NEXT I
430 RETURN
```

At Lines 30 to 160, gross and net scores are calculated when data are input from the keyboard, and these values are stored in the arrays.
At Lines 170 to 260, input data can be corrected.
Final results are displayed at Line 280.
Lines 300 to 430 are the subroutine for displaying results. The INKEY$ function in Line 370 reads one character input from the keyboard. When there is no input from the keyboard, null (CHR$ (0): there is nothing) is returned. Accordingly, until a key is depressed, program execution remains stopped at Line 370. This also applies for Line 410. These lines are used so that after 3 lines are displayed

on the LCD, it will not scroll until a key is depressed.
Let's execute the program.

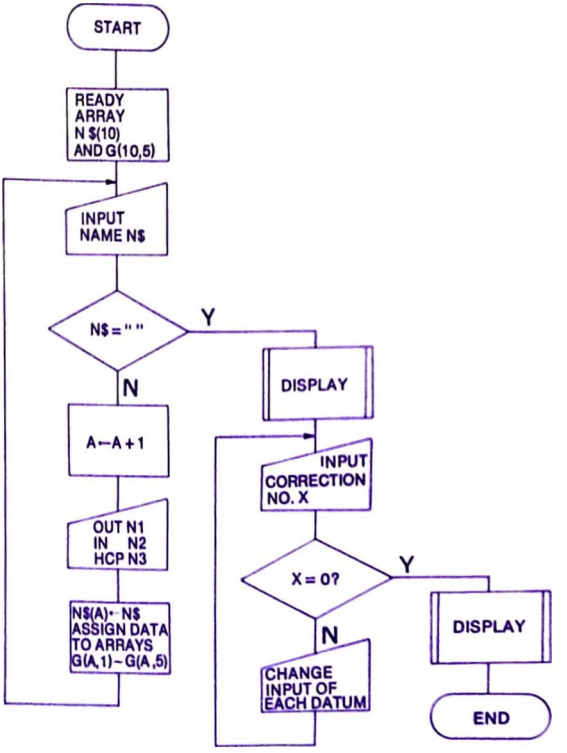
Data input

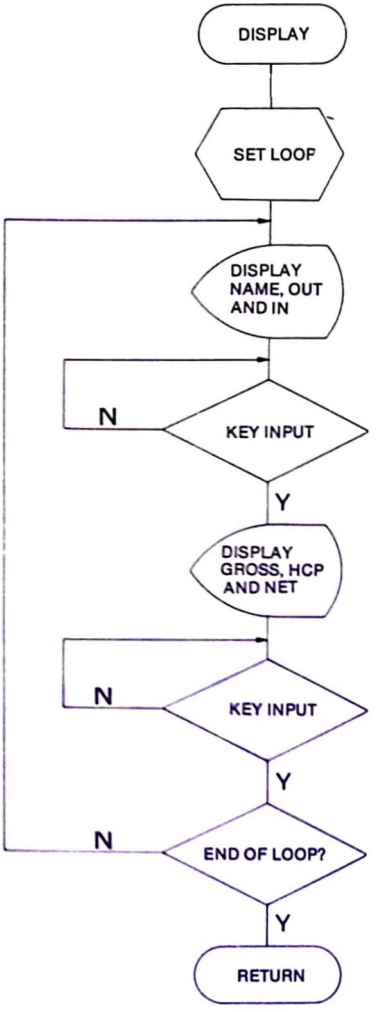
```
NAME?  CARL
OUT SCORE?  50
IN SCORE?  48
HCP?  5
NAME?  MARK
OUT SCORE?  55
IN SCORE?  52
HCP?  10
NAME?  ANN
OUT SCORE?  43
IN SCORE?  50
HCP  2
NAME? (←depress the RETURN key
CORRECTION NO.?  0
```

Final result (In case of no correction, 0 is input for "CORRECTION NO.")

```
NAME CARL
OUT  50
IN  48
GROSS 98 (←depress a key here)
HCP  5
NET 93 (←deress a key here)
2
NAME MARK
OUT  55
IN  52
GROSS  107 (←depress a key here)
HCP  10
NET  97 (←depress a key here)
3
```

NAME ANN
OUT 43
IN 50
GROSS 93 (←depress a key here)
HCP 2
NET 91 (←depress a key here)





New instruction

INKEY$

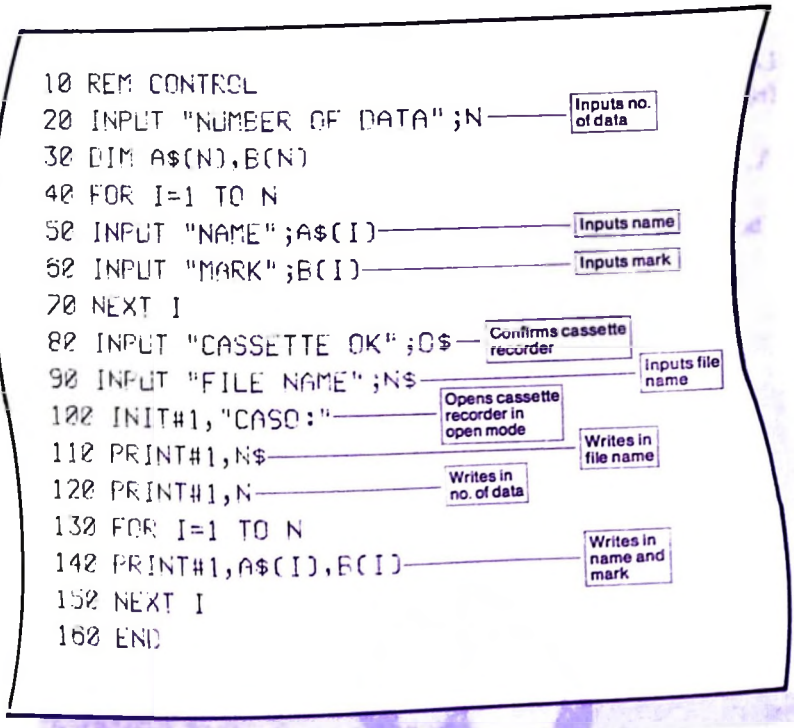
A function which reads one character from the keyboard.

3.4 Controlling records

Let's create a record file on a cassette tape or read a record file from a cassette tape.

A. How to create a record file

The following program is for creating a record file on a casstte tape.



First, the number of data N is read from the keyboard. Next, N number of names and marks are read from the keyboard, which are then written onto a cassette tape.
Let's now look at the lines related to creating the cassette file.

100 INIT #1, "CASO:"

A data file is opened in the cassette tape with the output mode. The baud rate is preset to 1200 baud and transmission mode is preset to B.

110 PRINT #1, N$
120 PRINT #1, N

The contents of the variable N$ (i.e. file name, let's call it MARKS), and the number of data N are written into the opened data file.

140 PRINT #1, A$ (I), B (I)

In the same manner as above, names and marks are written into the MARKS file.
The general procedure for creating a cassette file is as follows:

INIT #n, "CASO: <file name>"	Open
PRINT #n	Write

Note: <file name> is a dummy, and is not written onto the cassette tape. Therefore, it can be omitted. Accordingly, the actual file name (it is MARK in the sample program) is written at the beginning of a file, and is used to check a file when it is read later on.
Also, it is obvious that to write onto a cassette tape, the cassette recorder must be in the recording mode. For this reason, Line 80 asks if eveything is ready.

Let's execute the program in this way:

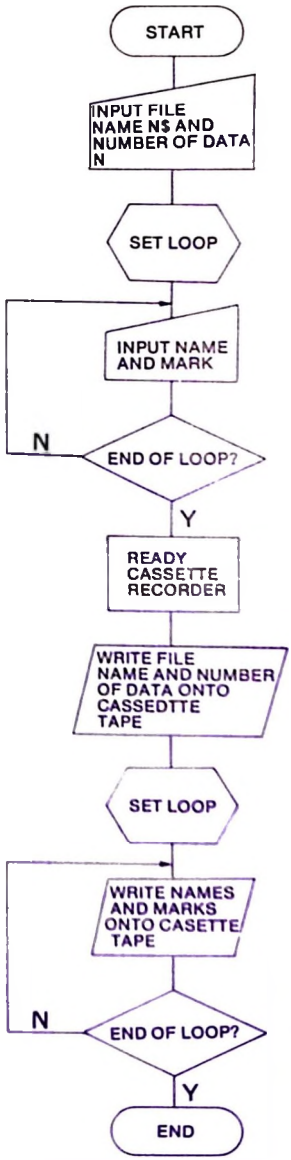
```
NUMBER OF DATA  4
NAME?  BOB
MARK?  76
NAME?  JEAN
MARK?  32
NAME?  ANN
MARK?  90
NAME?  RICHARD
MARK?  55
CASSETTE OK?.....
FILE NAME? MARKS
```

Put the cassette recorder into the recording mode, and depress a key.

MARKS file is created as follows, on the cassette tape.

MARKS

4	BOB	76	JEAN	32	ANN	90	RICHARD	55



New Instructions

INIT #n Opens a file.

PRINT #n Writes into a file.

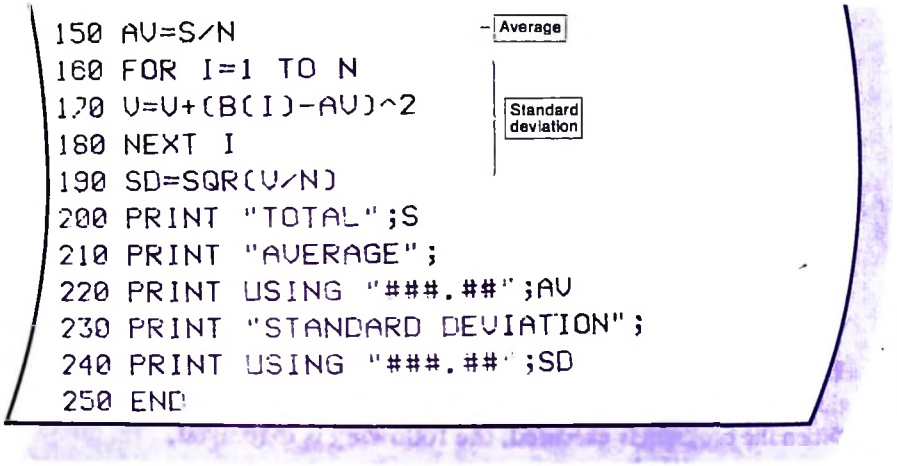
B. Calculating statistics from a record

The following program reads the data from the previously created cassette file, and then calculates the total, the average, and the standard deviatin of these data.
The expressions for calculating the total, the average, and the standard deviation are as follows:

n = number of students
$x_1, x_2, \ldots, xn$ = students' marks
$S = x_1 + x_2 \ldots + xn$
$\bar{x} = S/n$
$V = (x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \ldots + (xn - \bar{x})^2$
$SD = \sqrt{V/n}$

```
10 REM READ CONTROL
20 INPUT "FILE NAME";N$
30 INPUT "TAPE REWIND OK";O$
40 PRINT "DEPRESS PLAY"
50 INIT#1,"CAS1:"
60 INPUT#1,F$
70 IF F$=N$ THEN PRINT "FOUND ";N$ ELSE
60
80 INPUT#1,N
90 DIM B(N)
100 FOR I=1 TO N
110 INPUT#1,A$,B(I)
120 S=S+B(I)
130 PRINT A$;" ";B(I)
140 NEXT I
```

Total



First, rewind the cassette tape, and put the cassette recorder into the playback mode. When MARKS is input to the file name F$, this program finds the file starting with "MARKS", and reads the names and the marks which are then displayed on the LCD.
Lines for reading data from the cassette tape are as follows:

```
20  INPUT "File name"; N$
50  INIT #1, "CASI:"
```

The data file of the cassette tape is opened with the input mode.

```
60  INPUT #1, F$
80  INPUT #1, N
110  INPUT #1, A$ , B (I)
```

Lines 60 and 70 search for the desired file. At Line 80, the number of data is read into the variable N. At Line 110, the names and the marks are read into the variables A$ and B, respectively

The general procedure for reading data from a cassette tape is as follows:
Note: As before, <file name> has no acutal meaning.

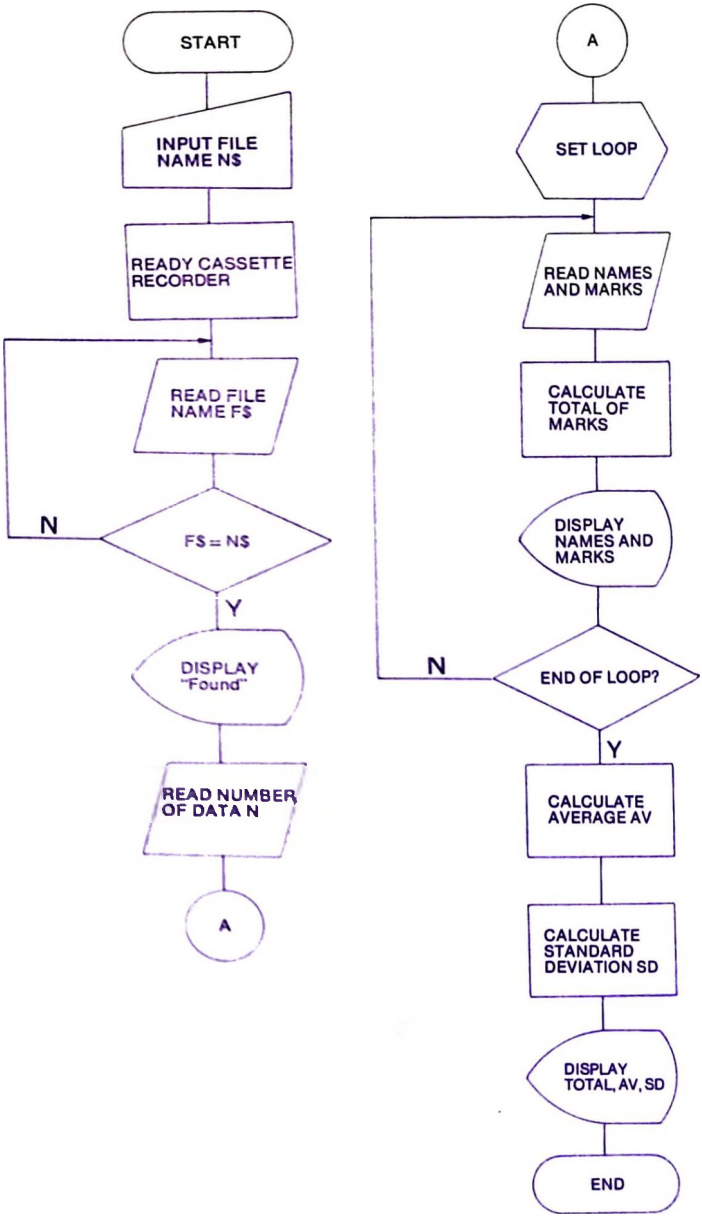
INIT #n, "CASI: <file name>"	Open
INPUT #n	Read

The USING instruction causes the average and the standard deviation to be displayed with 2 decimal places.
When the program is executed, the following is displayed.

```
BOB   76
EAN   32
NN    90
ICHARD  55
TOTAL  253
AVERAGE  63.25
STANDARD DEVIATION  21.92
```

New Instructions

INPUT #n Reads a file.
SQR Function for returning a square root of a value.


START
INPUT FILE NAME N$
READY CASSETTE RECORDER
READ FILE NAME F$
F$ = N$
N
Y
DISPLAY "Found"
READ NUMBER OF DATA N
A
A
SET LOOP
READ NAMES AND MARKS
CALCULATE TOTAL OF MARKS
DISPLAY NAMES AND MARKS
END OF LOOP?
N
Y
CALCULATE AVERAGE AV
CALCULATE STANDARD DEVIATION SD
DISPLAY TOTAL, AV, SD
END

Practical programs

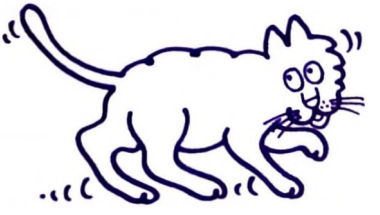
At first, creating practical programs might seem complicated, but it is often much simpler than creating games. When creating a practical program, just keep clear what to input and what to output, and the rest will come naturally as long as you try to keep it simple. Create a program like you would a flowchart, with correct flow of calculations, displays, memorizations, and decisions.

On the other hand, though games are treated lightly, they require a very high degree of programming skill. For one thing, what must be output is "fun", and there is nothing more ambiguous and personal as "fun". To make this clearly into a program is like performing acrobatics. However, the easy part of creating such a program is that there can be "few inaccuracies". In other words, it is all right if part way through a program the contents of the game change because the purpose has changed.

In the end, practical programs are hard to create, because it is difficult to create a program which has wide general application, and that there can never by any inaccuracies or mistakes. However, there is no more easier programming than creating a program which only you can use, and which is of only special use (only for your job). Anyhow, try creating a program.

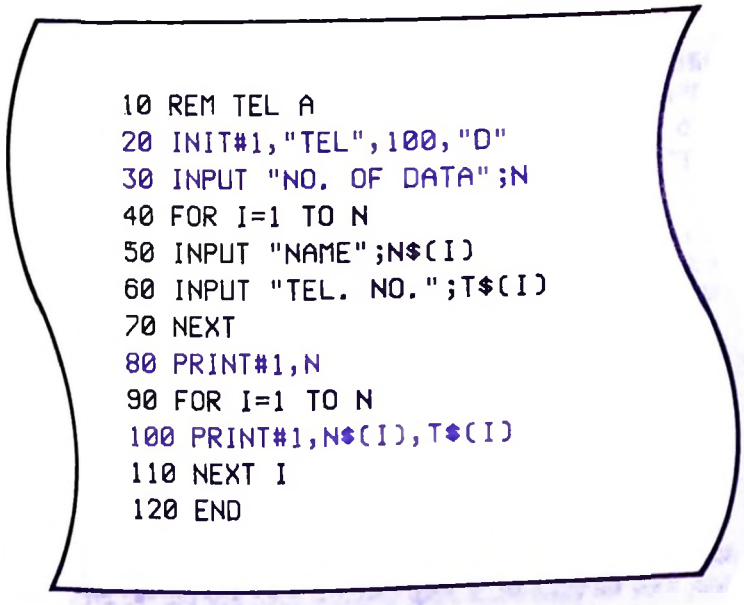
3.5 Telephone directory file

Data files can be created inside the RAM, as well as program files. However, only sequencial files can be created.
In this section, let's create programs for creating a telephone file and reading it.
When creating a file in the RAM, the size of the RAM file must be set with the FSET command. Here, set the RAM file to 1024 bytes (FSET 1024). If you have any difficulty setting the RAM file, refer back to "1.5 Memory of the X-07" of the User's Guide, or to "FSET" of the BASIC Reference Manual.



A. How to create a telephone directory file

The following program creates a telephone directory file in the RAM file.



```
10 REM TEL A
20 INIT#1,"TEL",100,"D"
30 INPUT "NO. OF DATA";N
40 FOR I=1 TO N
50 INPUT "NAME";N$(I)
60 INPUT "TEL. NO.";T$(I)
70 NEXT
80 PRINT#1,N
90 FOR I=1 TO N
100 PRINT#1,N$(I),T$(I)
110 NEXT I
120 END
```

Names and telephone numbers are read from the keyboard, which are then written into the RAM file to create the TEL file.
Let's look at one line which deals with the RAM file.

20 INIT #1, "TEL", 100, "D"

Line 20 opens the file. In other words, a RAM file with file name TEL, file size of 100 bytes (for approximately 100 characters), and of data file type is reserved. When dealing with data for many people, a

larger file size is set.

```
30  INPUT "NO OF DATA"; N
```

The number of people to be handled is set at Line 30, and read into the variable N. After Line 30, each line is executed N times. Here, if data of more than 10 people are to be handled, array varaibles must be declared as after Line 30 with DIM N$ (N), T$ (N).

```
80  PRINT #1, N
100  PRINT #1, N$(I), T$(I)
```

Line 80 writes the number of data (N) at the beginning of the TEL flie. Line 100 writes in consecutively the names N$ (I) and the telephone numbers into the TEL file. #1 denotes the file number, and it must be same as the file number specified by the INIT # statement.
The general procedure for creating a data file in the RAM file is as follows:

INIT # n "file name", number of bytes, "file type"	Open
PRINT #n	Write

Let's execute the program.

```
NO. OF DATA  4
NAME?  MARY
TEL. NO.?  987-4534
NAME?  GEORGE
TEL. NO.?  345-5432
NAME?  ANITA
TEL. NO.?  423-1298
NAME?  SAM
TEL.  NO.?  110-1111
```

The TEL file is set as follows:

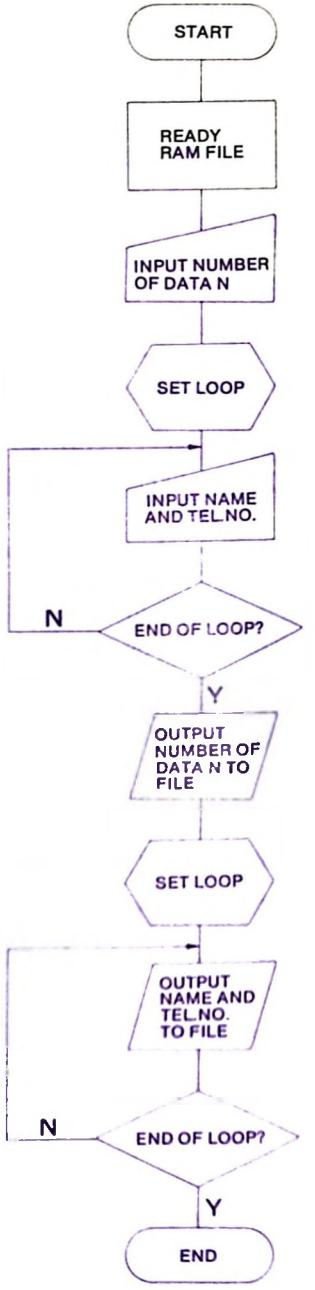
TEL

```
4 MARY 987-4534 GEORGE

345-5432 ANITA 423-1298

SAM 110-1111
```

Please make sure that the TEL file has been created in the RAM file with the DIR command.



B. How to read the telephone directory file

Let's now write a program to read the previously created telephone directory file.

```
10 REM TEL B
20 INIT#1,"TEL"
30 INPUT#1,N
40 FOR I=1 TO N
50 INPUT#1,N$,T$
60 PRINT N$," ";T$
70 IF INKEY$="" THEN 70
80 NEXT I
90 END
```

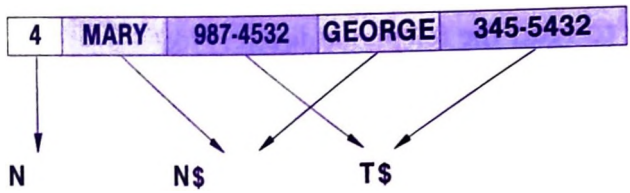
This opens the TEL RAM file.

30 INPUT #1, N
50 INPUT #1, N$, T$

A Line 30, the number of data in the TEL file is read, and, at Line 50, the names and the telephone numbers are read into the variables N$ and T$.

70 IF INKEY$ = "" THEN 70

Line 70 is set so that the next display is not shown until a key is depressed.

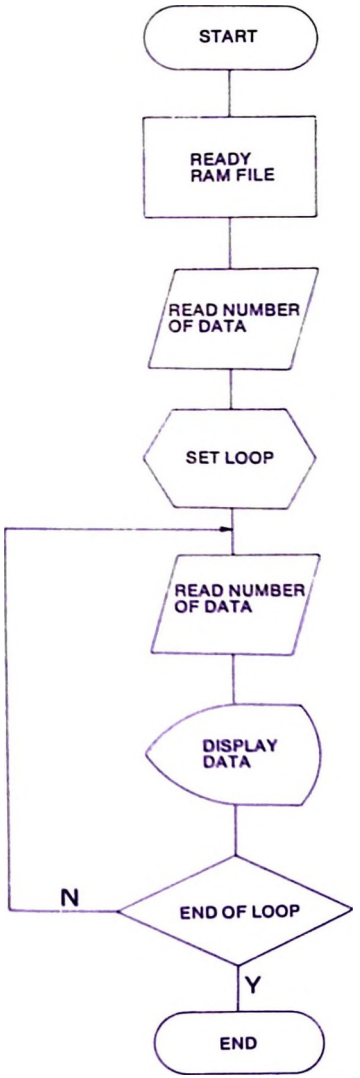


The general procedure for reading a file is as follows:

INIT #n "file name"	Open
INPUT #n	Read

The following is displayed when the program is executed. Please depress a key after one line is displayed.

MARY 987-4534
GEORGE 345-5432
ANITA 432-1298
SAM 110-1111


START
READY RAM FILE
READ NUMBER OF DATA
SET LOOP
READ NUMBER OF DATA
DISPLAY DATA
N
END OF LOOP
Y
END

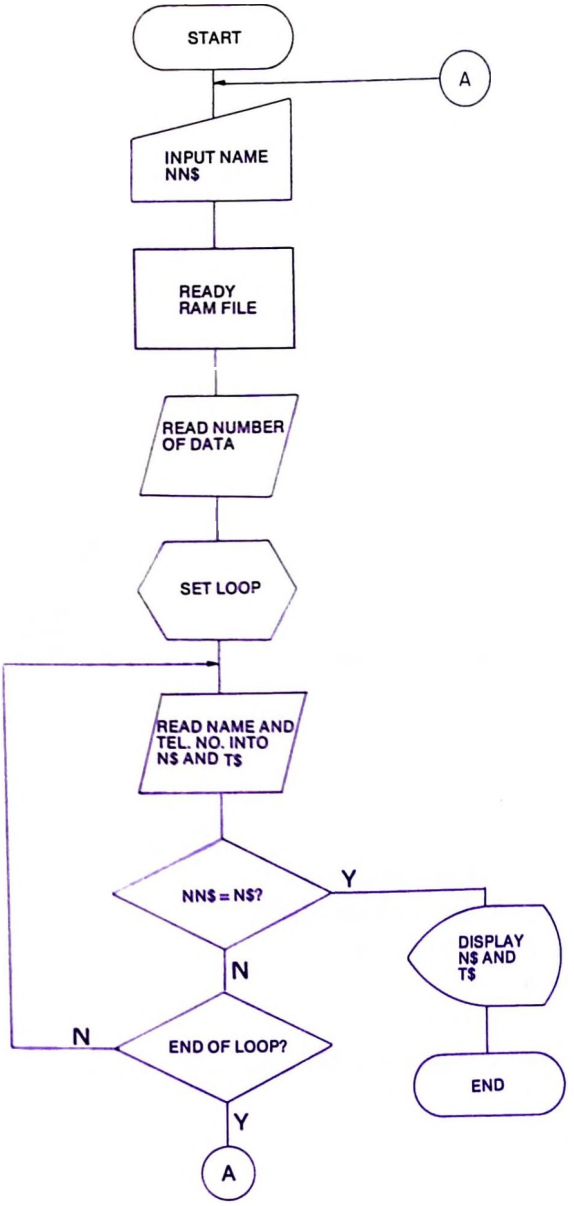
C. How to retreive data from the telephone directory file

The following program lets you display a telephone number when a name is input from the keyboard.

```
10 REM TEL C
20 INPUT "NAME";NN$
30 INIT#1,"TEL"
40 INPUT#1,N
50 FOR I=1 TO N
60 INPUT#1,N$,T$
70 IF N$=NN$ THEN 100
80 NEXT I
90 GOTO 20
100 PRINT N$;" ";T$
110 END
```

Following is displayed when the program is executed.

```
NAME? GEORGE
GEORGE 345-5432
```


START
A
INPUT NAME
NN$
READY
RAM FILE
READ NUMBER
OF DATA
SET LOOP
READ NAME AND
TEL. NO. INTO
N$ AND T$
NN$ = N$?
Y
N
DISPLAY
N$ AND
T$
END OF LOOP?
N
END
Y
A

3.6 Inventory control

As another example of a program which uses the RAM file, let's create an inventory control program with the following features:

- Adds new inventory data.
- Updates the number of goods in the existing inventory.

The program is as follows:

```
10 REM INVENTORY CONTROL
20 ON ERROR GOTO 500
30 DIM A$(2,100)
40 INIT#1,"INVENTORY",100,"D"
50 INPUT#1,N
60 IF N=0 THEN 100
70 FOR I=1 TO N
80 INPUT#1,A$(0,I),A$(1,I),A$(2,I)
90 NEXT I
100 PRINT "1 ADD"
110 PRINT "2 UPDATE"
120 PRINT "0 END"
130 INPUT F
140 IF F=1 THEN GOSUB 220
150 IF F=2 THEN GOSUB 300
160 IF F<>0 THEN 100
170 PRINT#1,N
180 FOR I=1 TO N
190 PRINT#1,A$(0,I),A$(1,I),A$(2,I)
200 NEXT I
210 END
220 REM ADD
```

Necessary when there is no file

Reads no of product data

Reads product data

Displays menu select job

Writes in file

```
230 N=N+1
240 INPUT "PRODUCT CODE";A$(0,N)
250 INPUT "PRODUCT NAME";A$(1,N)
260 INPUT "QUANTITY";A$(2,N)
270 RETURN
300 REM UPDATE
310 INPUT "PRODUCT CODE";SC$
320 FOR I=1 TO N
330 IF A$(0,I)=SC$ THEN GOSUB 380:I=N:M=
1
340 NEXT I
350 IF M=1 THEN 370
360 PRINT "NOT FOUND"
370 M=0:RETURN
380 PRINT A$(0,I),A$(1,I),A$(2,I)
390 INPUT "INCREASE(+)DECREASE(-) QUANTI
TY";L
400 X=VAL(A$(2,I))
410 X=X+L
420 A$(2,I)=STR$(X)
430 RETURN
500 REM ERROR RECOVERY
510 IF ERR<>22 THEN END
520 PRINT "NO FILE"
530 PRINT "INPUT DATA (1 ADD)"
540 FOR I=0 TO 500:NEXT
550 RESUME 100
```

Process when there are no data

This program is executed as follows.

1. Select the function you want when the following is displayed.

```
1 ADD
2 UPDATE
0 END
?
```

   1 To add new data to the inventory file.
   2 To update the number of goods.
   0 To terminate the program.

2. To add new data, input the product code, the product name, and the quantity.

   PRODUCT CODE?
   PRODUCT NAME?
   QUANTITY?

3. To update the quantity of a product, input the code of the product to be updated.

   PRODUCT CODE?

   Next, add to or substract from the existing quantity. When subtracting, place a "-" before the number.

   INCREASE (+) DECREASE (−) QUANTITY?

4. IO Error occures when the computer tries to read data when there are no data to be read. Here, in this case only, you must show that there are no data after Line 500, and transfer the control to the menu screen. From there, select "1", and input data. This error recovery procedure is necessary, for example, when you are executing this program for the first time.

The INVENTORY file is created in the RAM, and is structured as follows:

The number of products in the INVENTORY file is input at the beginning, followed by the product data.
Let's execute the program.

```
1  ADD
2  UPDATE
0  END
? 1
PRODUCT CODE?  10
PRODUCT NAME?  PEN
QUANTITY?  30
```

Add

```
1  ADD
2  UPDATE
0  END
? 1
PRODUCT CODE?  20
PRODUCT NAME?  NOTE
QUANTITY?  30
```

Add

```
1  ADD
2  UPDATE
0  END
? 2
PRODUCT CODE?  10
10  PEN
30
INCREASE (+) DECREASE (-) QUANTITY?  5
```

Update

Increase

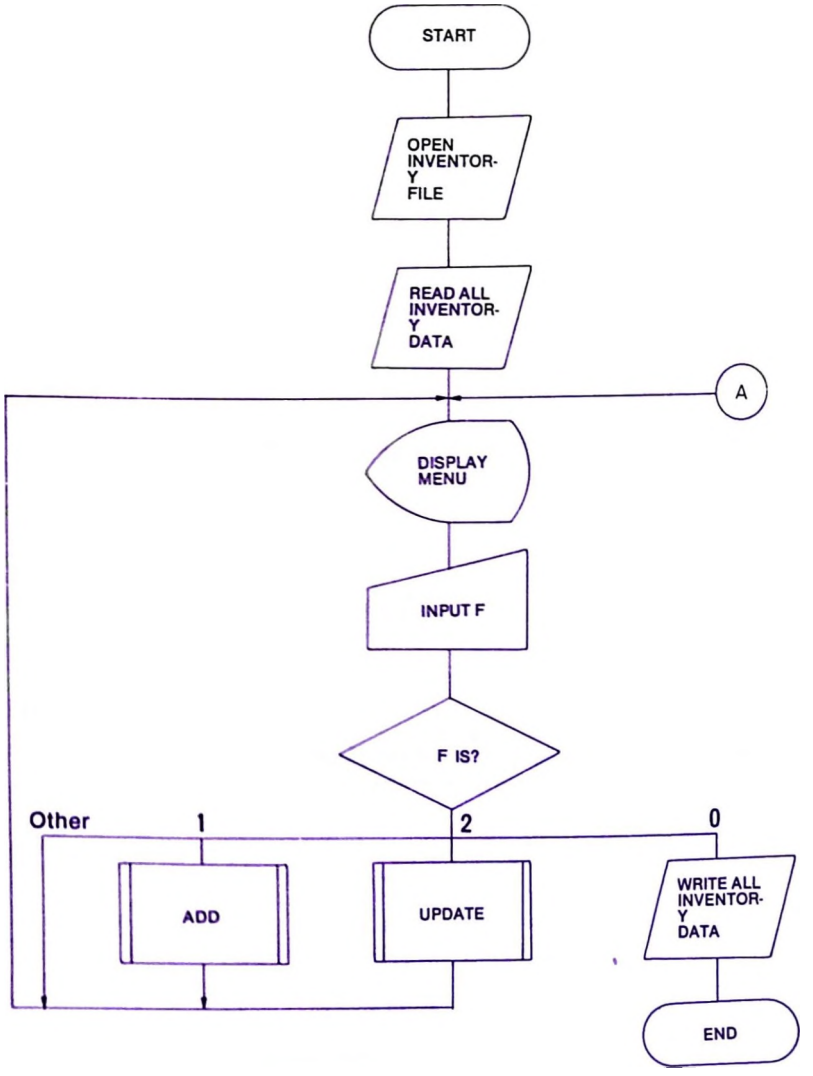
Update

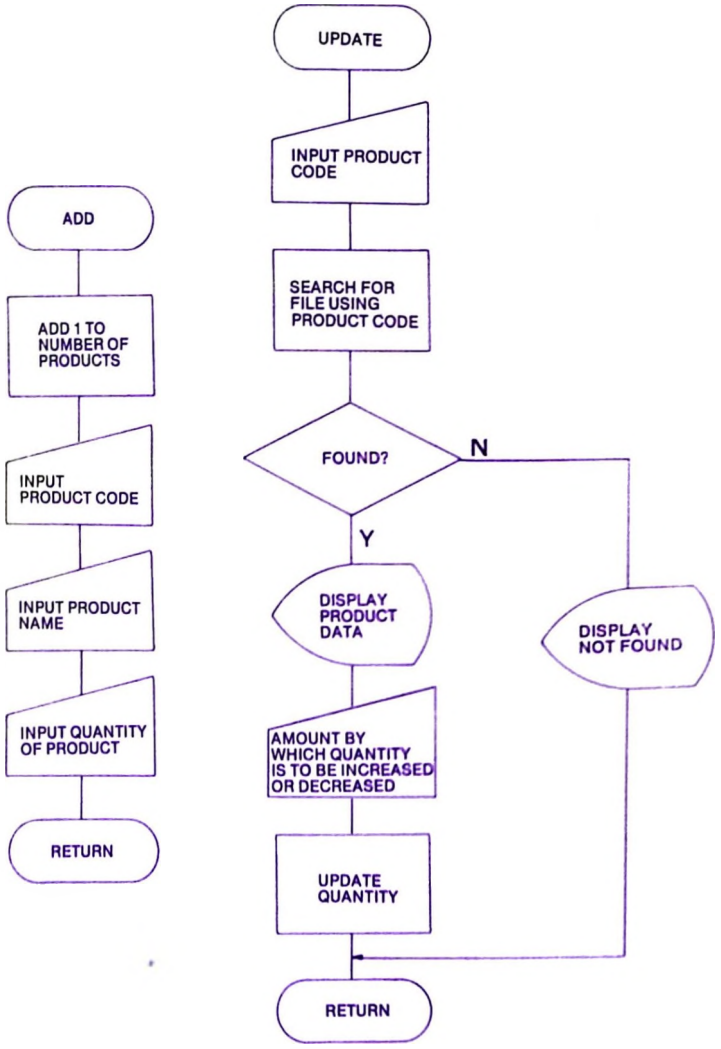
```
1  ADD
2  UPDATE
0  END
? 2
PRODUCT CODE?   10
10   PEN
35
INCREASE (+) DECREASE (–) QUANTITY?   –10
```

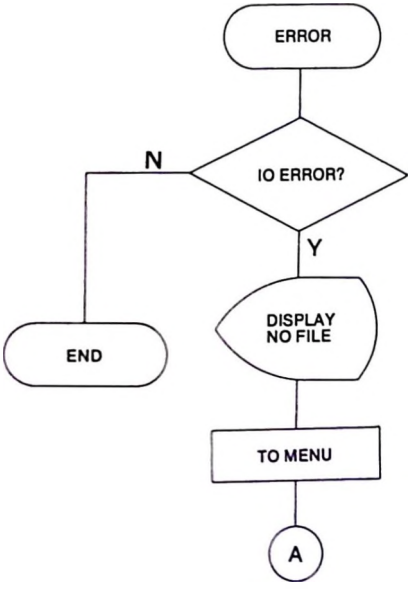
Decrease

End

```
1  ADD
2  UPDATE
0  END
? 0
```


START
OPEN INVENTOR-Y FILE
READ ALL INVENTOR-Y DATA
A
DISPLAY MENU
INPUT F
F IS?
Other
1
2
0
ADD
UPDATE
WRITE ALL INVENTOR-Y DATA
END


ADD
ADD 1 TO NUMBER OF PRODUCTS
INPUT PRODUCT CODE
INPUT PRODUCT NAME
INPUT QUANTITY OF PRODUCT
RETURN
UPDATE
INPUT PRODUCT CODE
SEARCH FOR FILE USING PRODUCT CODE
FOUND?
N
Y
DISPLAY PRODUCT DATA
DISPLAY NOT FOUND
AMOUNT BY WHICH QUANTITY IS TO BE INCREASED OR DECREASED
UPDATE QUANTITY
RETURN



New Instructions

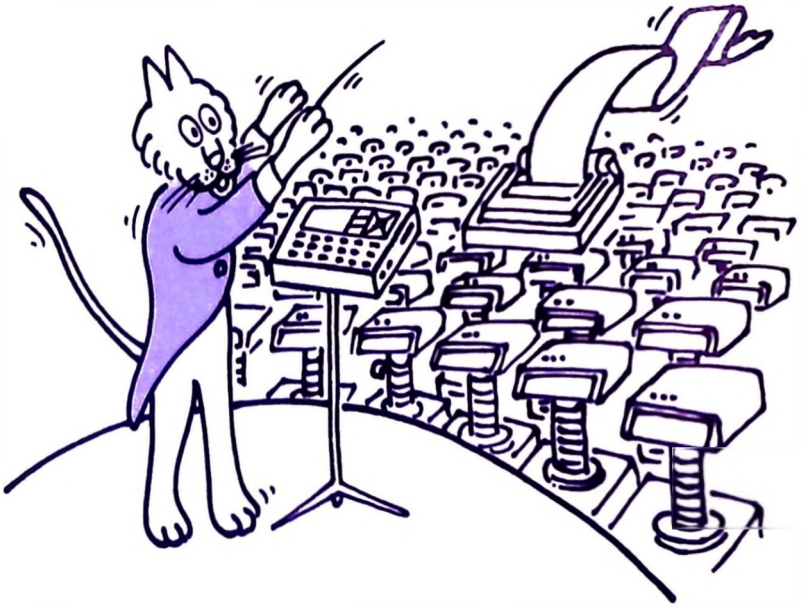
ON ERROR GOTO	Specifies error recovery subroutine.
RESUME	Returns from error recovery subroutine.
ERR	This is a reserved variable with error code (shows which error occurred).
STR$	This function converts a numeric value to a number string.
VAL	This function converts a number string to a numeric value.

Chapter 4
Programs which make the best use of the features of the X-07

The X-07 has the following features:

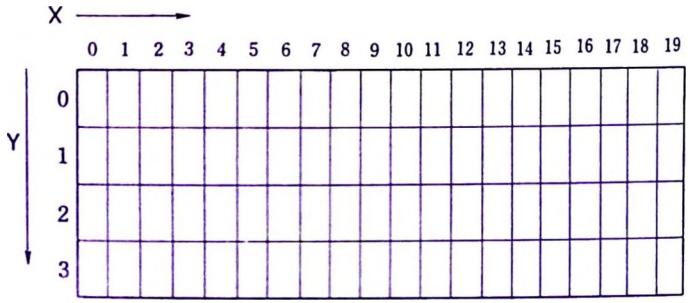
- **4 × 20 character display**
- **Speaker for playing back music**
- **32 × 120 dot graphic display**
- **Connection of a graphic printer**
- **Connection of an optical coupler**

This chapter introduces programs utilizing these features.



4.1 Character display

The LCD is 20 (units) × 4 (lines) large, and the coordinates are set as follows:



The screen may be small, but interesting programs can be created.

A. Digital timer with an alarm

The following program displays a digital timer on the LCD.

```
10 REM ALARM
20 CONSOLE@1:CLS
30 LOCATE 5,0:PRINT DATE$
40 LOCATE 5,2:PRINT TIME$
50 A$=INKEY$:I=I+1
60 IF I=10000 THEN I=0:SLEEP
70 IF A$<>"A" THEN 30
80 PRINT "CURRENT ALARM TIME",ALM$
90 INPUT "ALARM TIME(H,M)";B$,C$
100 ALM$="////"+B$+":"+C$
110 CLS
120 GOTO 30
```

Year, month, day, and day of week will be displayed on the LCD. These items are advanced with passage of time. Also, when the A key is depressed, the alarm can be set, and the alarm will sound for 1 minute at the specified time.

20 CONSOLE @ 1: CLS

The CONSOLE @ 1 statement turns ON the alarm, and the CLS statement clears the LCD.

30 LOCATE 5,0: PRINT DATE$

The LOCATE statement moves the cursor to the specified coordinates. In this case, it is moved to the (5,0) coordinates. The next instruction word "PRINT" displays characters starting at the cursor position. The variable DATE$ contains the date. Therefore, the date is displayed starting at the (5,0) coordinates.

```
40  LOCATE 5,2: PRINT TIME$
```

This is similar to the above. The variable TIME$ contains the actual time. Therefore, the actual time is displayed starting at the (5,2) coordinates.

```
50  A$=INKEY$:I=I+1
70  IF A$<>"A" THEN 30
```

The INKEY$ function reads the character of a depressed key. Unlike the INPUT statement, the value of a key is read without having to depress the RETURN key. The Lines 50 and 70, reads the value of the depressed key, and then checks whether it is [A] or not.

```
60  IF I=10000 THEN I=0: SLEEP
```

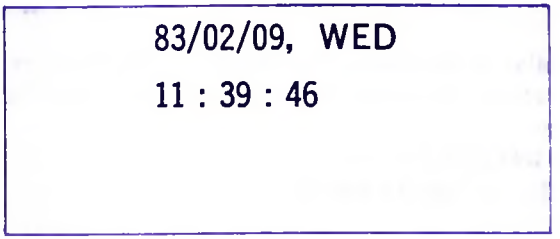
At line 60, if the result of the counter, which is incremented by 1 in the latter half of Line 50, is greater than 10000, the counter is reset to 0, and the SLEEP statement is executed. This line automatically turns off the power of the X-07 a little after the program is RUN, and automatically reopens the program when the power of the X-07 is turned on.

```
80  PRINT "CURRENT ALARM TIME", ALM$
```

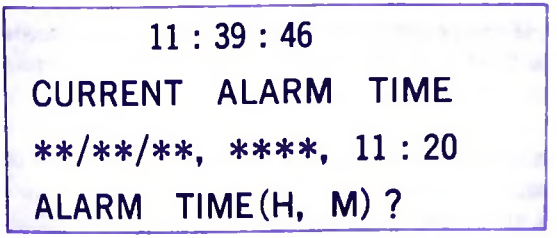
The variable ALM$ contains the time to which the alarm is currently set, and the time is displayed when Line 80 is executed.

```
90  INPUT "ALARM TIME (H,M)"; B$, C$
100  ALM$="////"+B$+":"+C$
```

At Line 90, the hours and minutes are input, and at Line 100, they are read into the varaible ALM$ as the new alarm time. When the alarm time is reached, the alarm will sound.
When the program is executed, the following is displayed:



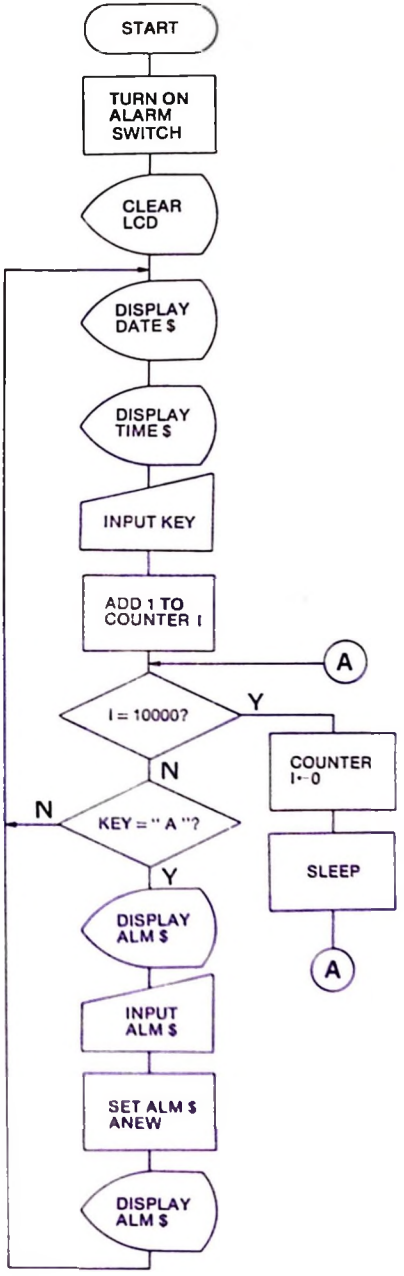
The date and the time will be advanced with passage of time. When the [A] key is depressed at a desired time, the following is displayed:



Here, for example, if

15,6

is typed, the alarm will sound at 15:06.



New instructions

LOCATE X, Y
Moves the cursor to (X, Y).

CLS
Clears the LCD.

CONSOLE @
Sets the alarm switch, etc.

SLEEP
Turns off the power of the 7 X-07 while retaining the present status.

DATE$
This variable is set with date.

TIME$
This variable is set with time.

ALM$
This variable is set with alarm time.

B. Memory dump

Memory dump means reading and displaying the contents of memory.

```
10 REM MEMORY DUMP
20 INPUT "START,FINAL";X,Y
30 AD$=HEX$(X)          ---- Converts address to hexadecimal number
40 PRINT AD$;" ";       ---- Displays address
50 A=PEEK(X)            ---- Reads memory data
60 A$=HEX$(A)           ---- Converts data to hexadecimal numbers
70 PRINT A$;" ";        ---- Displays data (hexadecimal)
80 IF A<32 THEN A=32
90 B$=CHR$(A)           ---- Converts data to characters
100 PRINT B$            ---- Displays data (characters)
110 X=X+1
120 IF X<=Y THEN 30
130 END
```

The contents of the specified memory range are displayed in the following format:

Address (hexadecimal) Datum (hexadecimal) Datum (character)

20 INPUT "START, FINAL"; X, Y

At Line 20, the starting address and the final address of memory to be dumpped are input from the keyboard. When variables are separated by comma(s), many data can be input with just one INPUT statement.

30 AD$ = HEX$ (X)

The HEX$ function converts a decimal number to a hexadecimal character string. Here, the address is changed to its hexadecimal form.

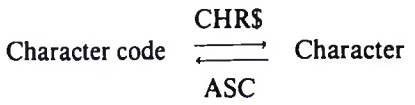
50 A = PEEK (X)

The PEEK function reads contents (1 byte) of memory. The read contents are stored in the variable A. To write data into memory, instead of reading data from memory (the PEEK function), the POKE function is used.

80 IF A < 32 THEN A = 32
90 B$ = CHR$ (A)

The CHR$ function converts a character code into a character. At Line 80, when the variable A is less than 32, the variable B$ is set to blank.

This is because character codes less than 32 are control codes, and a desired character might not be displayed on the LCD. Here, the read contents of memory are converted into characters. To convert a character into its character code, instead of vice versa (the CHR$ function), the ASC function is used.




```
40  PRINT AD$;" ";
70  PRINT A$;" ";
100  PRINT B$
```

These lines display each address (hexadecimal number) and its contents (hexadecimal number and character string). Please note that the Lines 40 and 80 end with a semi-colon (;). When a line ends with a semi-colon, carriage return/line feed does not occur. Accordingly, the result of a PRINT statement following such a PRINT statement is displayed on the same line. Therefore, in this case, contents of variables AD$, A$, and B$ are displayed on the same line.
The following is displayed when the program is executed.

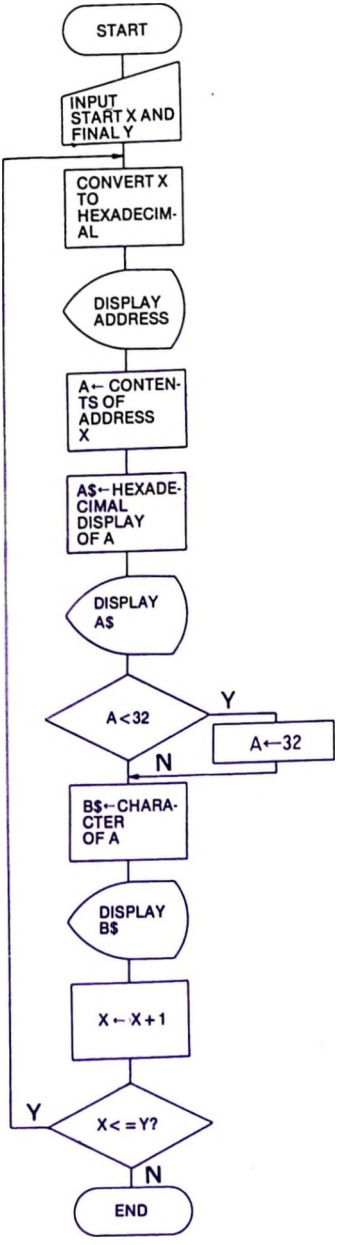
```
Input in decimal form

START, FINAL ? 10, 20
A   F5
B   64  4
C   0
D   0
E   0
F   0
10  C3  テ
11  37  7
12   F5
13  0
14  0
```

Input in hexadecimal form

START, FINAL? &H10, &H20

```
10  C3  テ
11  37  7
12  F5
13  0
14  0
15  0
16  3
17  0
18  C3  テ
19  C7  ヌ
1A  C9  ノ
1B  3A  :
1C  0
1D  0
1E  0
1F  0
20  C3  テ
```



New instructions

HEX$ (n)	Converts a decimal number n into a hexadecimal number.
PEEK (A)	Reads the contents of address A.
POLE A, n	Writes n into address A.
CHR$ (C)	Converts character code C into a character.
ACS ("A")	Converts character "A" into a character code.

Identity of a computer

The heart of a computer is a 1 chip LSI called a "CPU". This chip is so small that it can be placed on the tip of a finger. This chip is created so that the voltages of several pins set the voltages of other pins. These pins alone do not perform anything, so a keyboard is connected, and when a key is depressed, voltages are applied to the pins corresponding to the depressed key. Next, a Liquid Crystal Display (LCD) is connected so that the resulting output voltages are combined and displayed. Just think of a LCD as an intricate connection of miniature light bulbs.

Next, a chip which temporarily stores the voltages generated by depressing a key is added. (The existence/nonexistence of voltages are placed in order.) Later, a circuit is added to decide what to do according to the combination of depressed keys. And, all these parts make up the body of the X-07.

The X-07 is much more complicated, and it would take a whole book to correctly explain it. However, the above simple explanation does apply to any computer. Even by knowing only what was described above, it is easier to understand such technical phrase as "extension of peripheral devices". For example, a printer and a RS-232C circuit are "addition units for extending the functions of the CPU", which react to the signals from the CPU and the pins of group of ICs which support the CPU.

C. Mirror image

This program shows a mirror image display.

```
10 REM REVERSE
20 CLEAR 200:DIM A$(19,2)
30 FOR Y=0 TO 2
40 FOR X=0 TO 19
50 A=SCREEN(X,Y)
60 A$(X,Y)=CHR$(A)
70 NEXT X
80 NEXT Y
90 CLS
100 FOR Y=0 TO 2
110 FOR X=19 TO 0 STEP -1
120 PRINT A$(X,Y);
130 NEXT X
140 NEXT Y
150 IF INKEY$="" THEN 150
160 GOTO 30
```

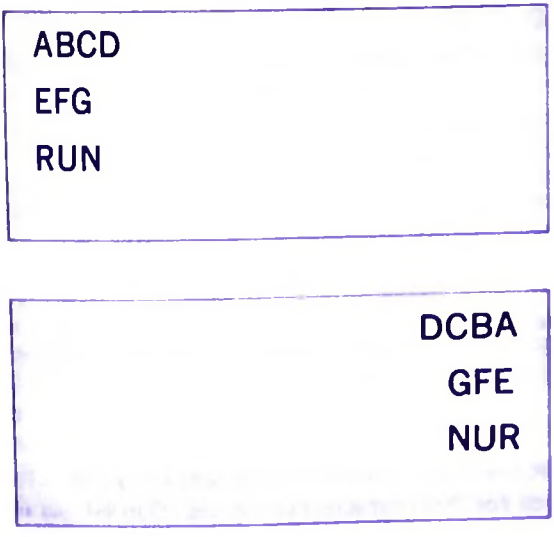
Reads lines 0 to 2 of LCD into A$

Display is reversed

The CLEAR statement is need when a large string area is to be used. Here, an area for 200 characters is reserved. Only 60 characters are assigned, one at a time, to the array variable A$; therefore, you would think that such a large area need not be reserved. However, an area is need for a string operation (Line 60), and a large area allows for speedier execution.

```
50  A = SCREEN (X, Y)
60  A$ (X, Y) = CHR $ (A)
```

The SCREEN instruction reads the character code of the character at the specified coordinates on the LCD. At Line 60, the read character code is converted to a character, and set into the array variable A$. In other words, at Lines 30 to 80, the characters displayed on the 0th to 2nd lines are set into the array variable A$.
At lines 80 to 120, the characters set into the array variable A$ are displayed on the LCD with left and right reversed. The STEP-1 clause in Line 110, decreases the counter of the FOR ~NEXT loop by one each time the loop is executed.
When this program is executed, the following is displayed:

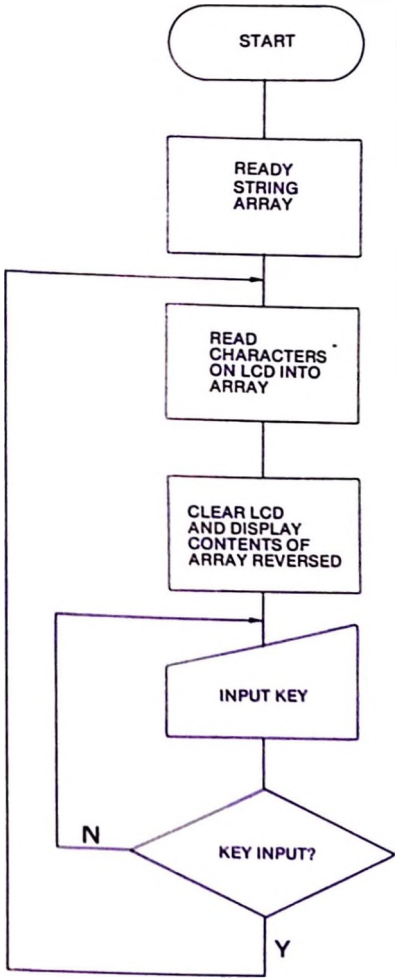


The program loops at Line 150.
The LCD is reversed every time a key is depressed.

New instructions

CLEAR
Reserves a string area, etc.

SCREEN (X, Y)
Reads character codes starting with the character at (X, Y).



D. Setting a character

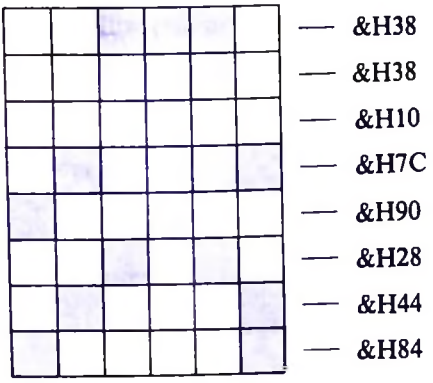
This program makes runners run across the LCD.

```
10 FONT$(130)="&H38,&H38,&H10,&H7C,&H90,
&H28,&H44,&H84"
20 CLS:D=RND(0)
30 FOR Y=0 TO 2
40 A(Y)=A(Y)+RND(1)*2
50 IF A(Y)>=18 THEN A(Y)=18:SW=1
60 LOCATE A(Y),Y
70 PRINT CHR$(130)
80 NEXT Y
90 IF SW=1 THEN END
100 FOR Y=0 TO 2
110 LOCATE A(Y),Y
120 PRINT " "
130 NEXT Y
140 GOTO 30
```

The Y-coordinates of the runners' are limited to 0, 1 and 2. The X-coordinates are changed by generated random numbers which are then set into A (0), A (1) and A (2). When a runner reaches X = 18, the game is terminated.

10 FONT$ (130) = "&H38, &H38, &H10, &H7C, &H90, &H28, &H44, &H84"

The FONT$ function resets a character. Here, the character code 130 is set to the character shown below.



20 CLS: D = RND (0)

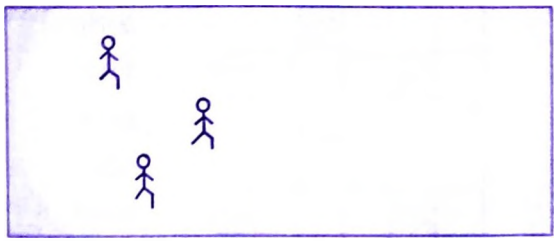
Here, the RND (0) generates new sequence of random numbers. When this is omitted, the runners will run in the same pattern no matter how many times this program is executed. To always generate a different sequence of random numbers, please remember the following points:

1. **Use D = RAND (0) near the beginning of the program.**
2. **Use RND (1) thereafter.**

70 PRINT CHR$ (130)

Line 70 displays the characters set at Line 10.
At Lines 30 to 80, the new coordinates of the 3 runners are set, and the characters set at Line 10 are displayed at these coordinates. At Line 40, random numbers are used to add numbers 0 to 2 to the X-coordinates of the 3 runners. At Line 50, when a runner reaches X-coordinate 18, the SW switch is set to 1. At Lines 60 and 70, the

characters set at Line 10 are displayed.
Line 90 checks for the end of the game. When the game is at an end, the program is terminated.
At Lines 100 to 130, the displayed characters are erased.
When this program is executed, 3 runners will race across the LCD.

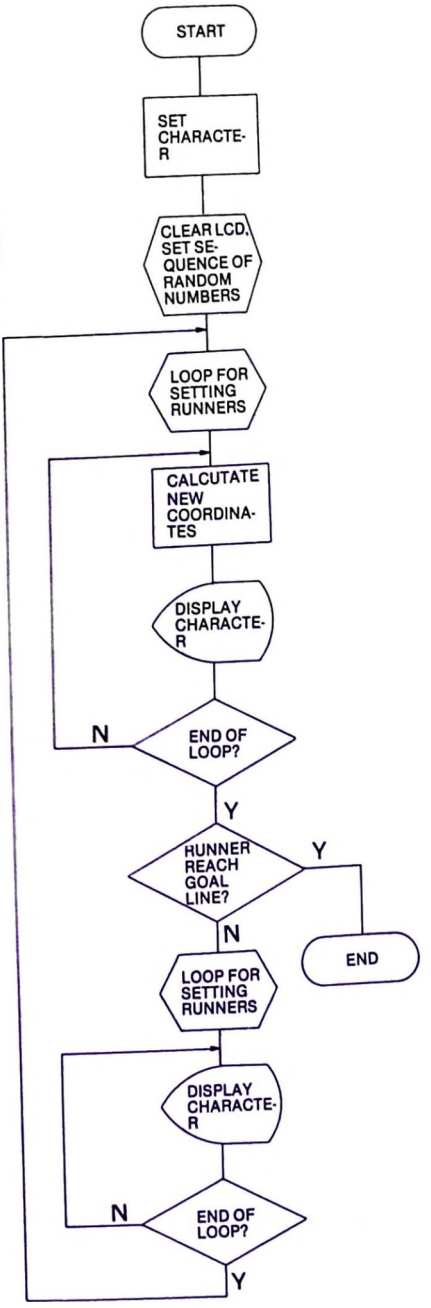


The goal line is 2 units from the left of the LCD. The race is over whenever a runner reaches the goal line. The winner will differ each time, so that the program can be enjoyed many times.

New instructions

FONT$ (character code)
Sets a character.

RND
Generates random numbers.



4.2 Playing back a musical selection

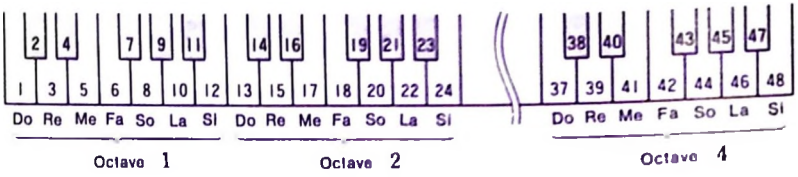
The BEEP statement can be used to output sound. Not only can a sound be output, but an entire musical selection can also be played back.
In this section, 2 programs which use the BEEP statement to play back music are introduced.

A. Twinkle, twinkle little star

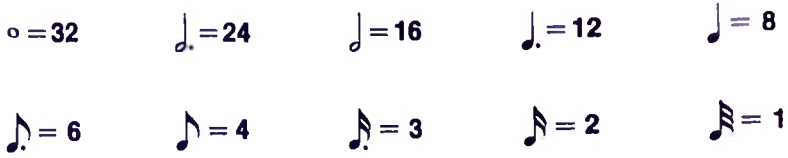
Let's play back "Twinkel, twinkle little star".
A tone and its length are set with the BEEP statement as follows:

BEEP I, J

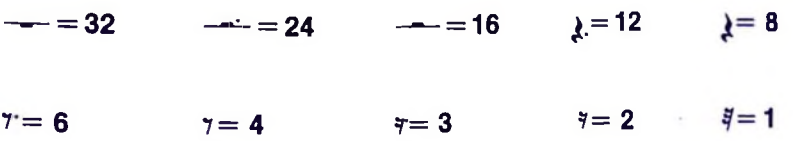
A tone is set with a number between 0 and 48 over a 4 octave range.



Also, when a tone is set to 0, no sound is output. Therefore, to specify a rest, set a tone to 0.
The length of a tone is from 0 to 255. For example, notes and their length can be set to as follows:

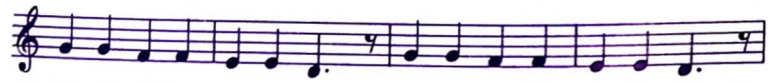
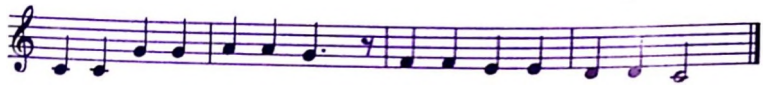


The length of rests can also be set.



When these values are doubled, the length of the notes and rests are doubled. In other words, the tempo of the music is halved.
The score of "Twinkle, twinkle little star" is shown on the next page.
The tone and the length of each note and rest are also shown.

Twinkle, twinkle little star

Tone	1	1	1	8	8	10	10	8	0	6	6	5	5	3	3	1	0
Length	♩	8	8	8	8	8	8	12	4	8	8	8	8	8	8	12	4

	1	8	8	6	6	5	5	3	0	8	8	6	6	5	5	3	0
	♩	8	8	8	8	8	8	12	4	8	8	8	8	8	8	12	4

	1	1	1	8	8	10	10	8	0	6	6	5	5	3	3	1
	♩	8	8	8	8	8	8	12	4	8	8	8	8	8	8	16

The program for "Twinkle, twinkle little star" is as follows:

```
10 DIM A(20),B(20),C(20)
20 DATA 1,1,8,8,10,10,8,0,6,6,5,5,3,3,1,
0
30 DATA 8,8,6,6,5,5,3,0,8,8,6,6,5,5,3,0
40 DATA 8,8,8,8,8,8,12,4,8,8,8,8,8,8,12,
4
50 FOR I=1 TO 16
60 READ A(I)
70 NEXT I
80 FOR I=1 TO 16
90 READ B(I)
100 NEXT I
110 FOR I=1 TO 16
120 READ C(I)
130 NEXT I
140 FOR I=1 TO 16
150 BEEP A(I),C(I)
160 NEXT I
170 FOR I=1 TO 16
180 BEEP B(I),C(I)
190 NEXT I
200 FOR I=1 TO 16
210 BEEP A(I),C(I)
220 NEXT I
230 END
```

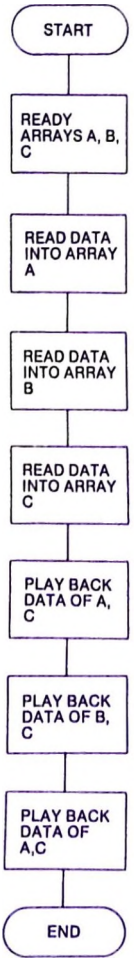
Reads DATA statements

Plays back second bar

Plays back third bar

Plays back second bar

The tones are specified by the DATA statements in Line 20 and Line 30, and the length of these tones are specified by the DATA statement in Line 40.
The BEEP statements in Line 150, 180, and 210 play back the first bar, the second bar, and the third bar, respectively.


START
READY ARRAYS A, B, C
READ DATA INTO ARRAY A
READ DATA INTO ARRAY B
READ DATA INTO ARRAY C
PLAY BACK DATA OF A, C
PLAY BACK DATA OF B, C
PLAY BACK DATA OF A,C
END

New instruction

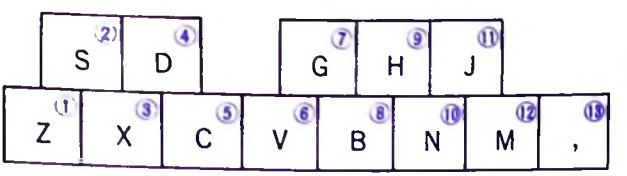
BEEP I, J Generates tone I for J length of time.

B. Keyboard

This program lets you use the keyboard like a piano.

```
10 REM PIANO
20 I$=INKEY$
30 IF I$="" THEN 20
40 I=INSTR("ZSXDCVGBHNJM,",I$)
50 BEEP I,4
60 GOTO 20
```

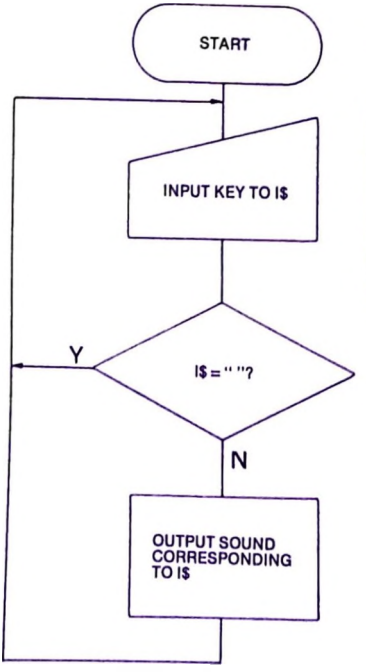
A sound is output when a key is depressed. A key and its corresponding tone are as follows:



Tone I = 1 ~ 13

At Line 20, a key is input. Line 30 checks whether a key was depressed, and if a key was not depressed, Line 20 is executed again. Until a key is depressed, the Line 20 ~ 30 loop is not terminated.
Line 40, checks which key was depressed, and the tone corresponding to the depressed key is set into the variable I. When a key which is not set with a tone is depressed, no sound is output, and the program execution returns to key input.

At Line 50, the sound corresponding to the depressed key is generated. The length of sound is fixed at 4. Before executing this program, it is best to turn off the key click sound (execute CONSOLE,,, 0).
Now, use the keyboard to play back a muscial selection.

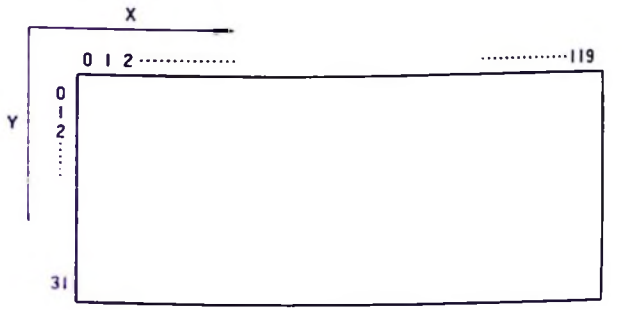


New instruction

INSTR (N$, I$)
Function which returns the position of I$ in N$.

4.3 Graphic display

You can use the X07-BASIC to draw circles, lines and dots on the LCD.
The LCD is a 120 (width) × 32 (length) dot graphic display, and the coordinates are set as follows:



A. Coordinate axes and dots

The program below draws coordinate axes, and draw dots at coordinates input from the keyboard.

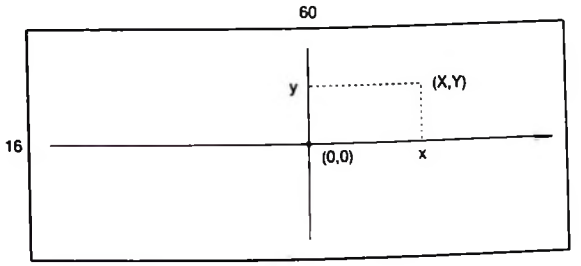
```
10 REM COORDINATES
20 CLS
30 LINE(0,16)-(120,16)
40 LINE(60,0)-(60,32)
50 INPUT"X,Y";X,Y
60 PSET(X+60,16-Y)
70 END
```

30 LINE (0, 16) - (120, 16)
40 LINE (60, 0) - (60, 32)

The LINE statement draws a line from one set of specified coordinates to another. Line 30 draws the X-axis from the (0, 16) coordinates to the (120, 16) coordinates. Line 40 draws the Y-axis from the (60, 0) coordinates to the (60, 32) coordinates.
At Line 50, coordinates of a dot corresponding to the above axes are input from the keyboard.

60 PSET (X + 60, 16-Y)

The PSET statement draws a dot at the specified coordinates. Coordinates input at Line 50 are relative to the drawn coordinate axes. However, as the PSET statemetn specifies only the absolute coordinates on the LCD, the input coordinates are converted to their absolute positions



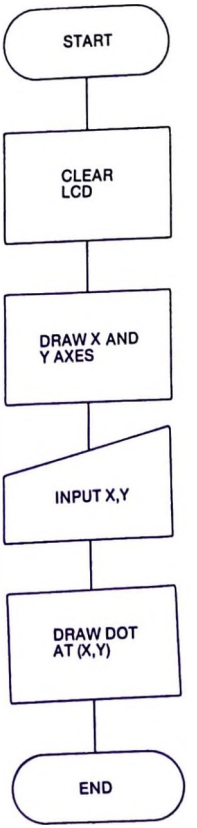
$X = x + 60$
$Y = 16 - y$

Try the program.

New instructions

PSET (X, Y)
Draws a dot at (X, Y).

LINE (X1, Y1) – (X2, Y2)
Draws a line from (X1, Y1) to (X2, Y2).



B. Moving dot

When the following program is executed, a dot moves all over the LCD.

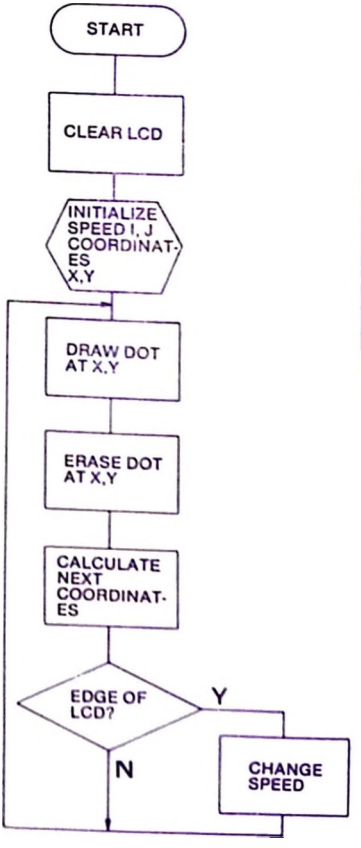
```
10 REM MOVE POINT
20 CLS
30 I=1:J=1
40 X=0:Y=0
50 PSET(X,Y)
60 FOR P=1 TO 5:NEXT P
70 PRESET(X,Y)
80 X=X+I:Y=Y+J
90 IF X=119 OR X=0 THEN I=-I
100 IF Y=31 OR Y=0 THEN J=-J
110 GOTO 50
```

Starting at the upper left corner, a dot moves across the LCD, and bounces off its edges, just like a billiard ball.
At Lines 50 to 60 dot is drawn using the PSET statement. Just the PSET statement could have been used, but the dot would have moved too fast for us to see it. Therefore, the PRESET statement was used.

70 PRESET (X, Y)

The PRESET statement erases the dot at the specified coordinates. In other words, this statement erases the dot drawn at Line 60.
Line 80 advances the coordinates of the dot.
Lines 90 and 100 check whether the dot has reached an edge of the

LCD. When it reaches an edge, the direction of the dot is changed. Now, try the program. You will see a ball bounce off the edges. When you want to terminate the program, depress the ON/BREAK key.

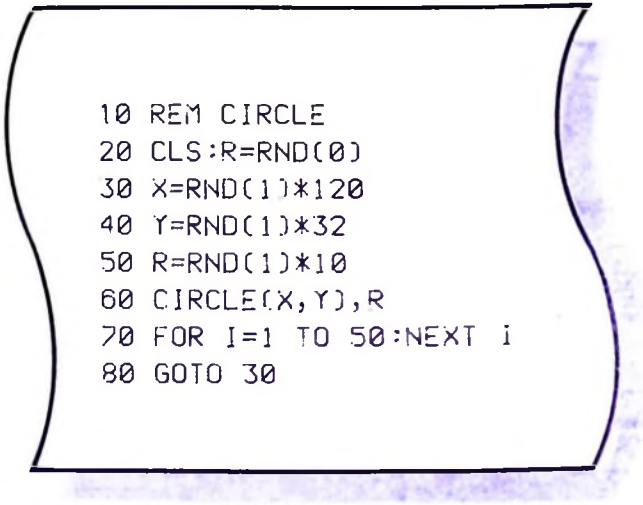


New instruction

PRESET (X, Y)
Erases the dot at (X, Y).

C. Drawing circles

The following program randomly draws circles.



```
10 REM CIRCLE
20 CLS:R=RND(0)
30 X=RND(1)*120
40 Y=RND(1)*32
50 R=RND(1)*10
60 CIRCLE(X,Y),R
70 FOR I=1 TO 50:NEXT I
80 GOTO 30
```

Lines 30 to 50 seek the X-coordinate, the Y-coordinate, and the radius R of a circle. The RND function generates random numbers between 0 and 1. Accordingly, the range of X, Y, and R are as follows:

$0 \leqq X < 120$
$0 \leqq Y < 32$
$0 \leqq R < 10$

60 CIRCLE (X, Y), R

The CIRCLE statement draws a circle with its center at the (X, Y) coordinates and radius of R.

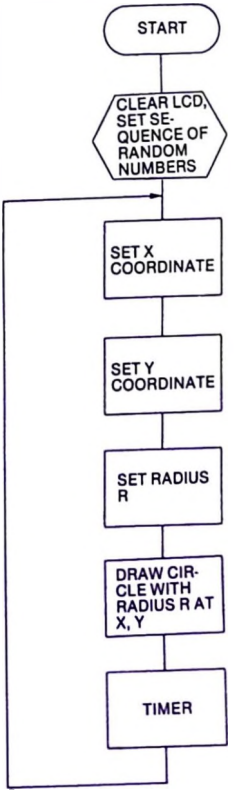
Line 70 is used to slow down the speed of execution, and is set to a reasonable period.

When this program is executed, circles of various sizes are drawn all over the LCD. To terminate this program, depress the ON/BREAK key.

New instruction

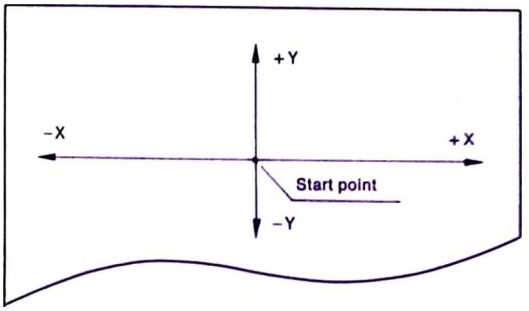
CIRCLE (X, Y), R

Draws a circle with centre at the (X, Y) coordinates and radius of R.



4.4 Printer

The Color Graphic Printer X-710 can be connected to the X-07. This printer can be used in both the text mode and the graphic mode. The text mode is used to print letters and numbers in the normal manner. In other words, to print programs, data, etc., use the text mode. When printing standard sized characters, up to 40 characters/line can be printed.
In the graphic mode, lines can be freely drawn, and characters can be printed in differenct colors, sizes and angles. When the printer is used in the graphic mode, it is set to the following coordinates.



Direction of paper feed
$-9990 \leqq X, Y \leqq 999$

The position of the start point can be set with a statement. Usually, the position of the pen at the left corner is the start point.

A. Printing characters

The following program prints all the characters set with character codes.

```
10 REM PRINTER
20 C=&H20
30 C$=CHR$(C)
40 LPRINT C$;
50 C=C+1
60 IF C<&HDF THEN 30
70 LPRINT
80 END
```

Character codes 20 (hexadecimal) to DF (hexadecimal) are converted into their corresponding characters and printed on the printer. In this case, the printer is in the text mode (this is the usual mode of the printer).

40 LPRINT C$;

The LPRINT statement is used to print data on the printer. This statement is similar to the PRINT statement. Here, characters are printed horizontally one at a time.
When this program is executed, the following is displayed:

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[¥]^_`abcdefghijklmno
pqrstuvwxyz{|}~⌧ 。「」、・ヲァィゥェォャュョッーアイウエオカキ
クケコサシスセソタチツテトナニヌネノハヒフヘホマミムメモヤユヨラリルレロワン゛゜
```

40 LPRINT "S"; I + 1

S sets the size of the characters. I changes from 0 to 3. Therefore, the size of the characters changes from S1 to S3.

50 LPRINT "C"; I

C sets the color of the characters. It changes to C0 (black), C1 (blue), C2 (green), and C3 (red).

60 LPRINT "Q"; I

Q sets the angle of the characters. Therefore, the characters are printed in series of 90° angles.

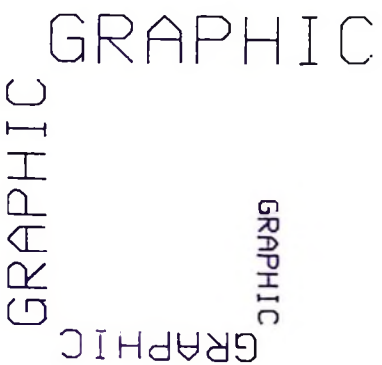
70 LPRINT "PGRAPHIC"

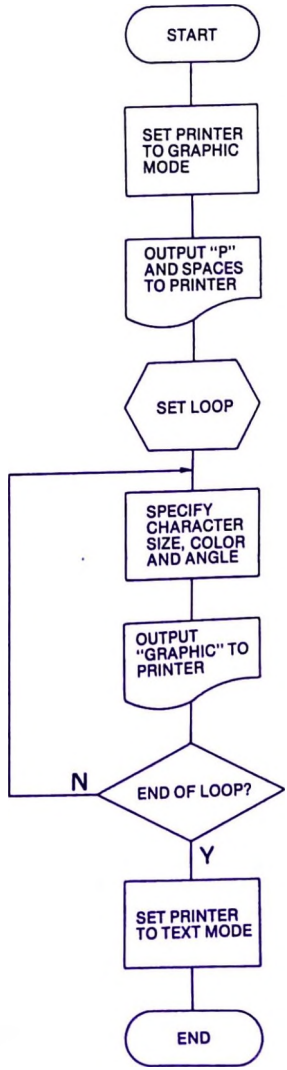
Here, "GRAPHIC" is printed.

90 LPRINT "S1"; CHR$ (17)

CHR$ (17) is DC1. When DC1 is sent to the printer, it is reset to the text mode. However, the status of S (character size) is retained. Therefore, S is set to S1 (40 characters/line) before the printer is reset to the text mode.

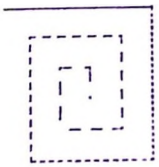
When this program is executed, the following is displayed:





C. Drawing a pattern

The following program draws the pattern shown below.



```
10 REM PGRAPHICS
20 LPRINT CHR$(18)
30 LPRINT "L0"
40 LPRINT "M200,0"
50 LPRINT "I"
60 FOR I=0 TO 9
70 READ X,Y
80 GOSUB 200
90 NEXT I
100 LPRINT "A"
110 END
120 DATA 100,0,100,-100,20,-100,20,-20,8
0,-20
130 DATA 80,-80,40,-80,40,-40,60,-40,60,
-60
200 REM PSUB
210 LPRINT "L";L
220 LPRINT "D";X;",";Y
230 L=L+1
240 RETURN
```

Lines 200 to 240 constitute a subroutine which draws a line from the current position of the pen to the (X, Y) coordinates.
This program has several statements for drawing a line. Let's examine them one by one.
First, at Line 20, the printer is set to the graphic mode.

30 LPRINT "L0"

When drawing a straight line, L is used to specify whether it will be a solid line or a dotted line. A solid line is drawn when L0 is sepcified, and a dotted line is drawn when L1 to L15 is specifed. Greater the number following L, wider the spaces between the dots.

40 LPRINT "M200, 0"

M raises the pen and moves it to the specifed coordinates. Here, the pen is moved to the (200, 0) coordinates.

50 LPRINT "I"

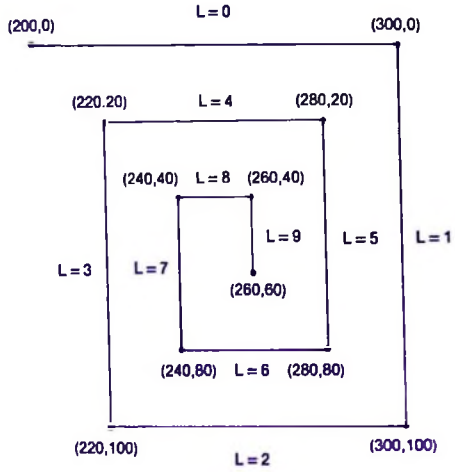
I sets the current pen position as the start point.

220 LPRINT "D"; X; ","; Y

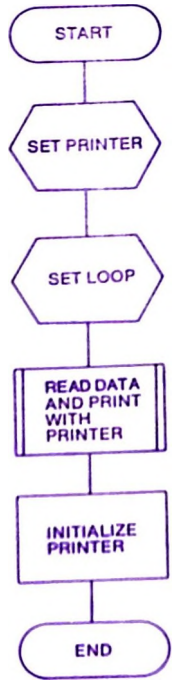
D draws a line from the current pen position to the (X, Y) coordinates.

100 LPRINT "A"

A raises the pen, returns it to the left corner, and then resets the printer to the text mode.
When this program is executed, the coordinates are changed as shown on the next page, and a line is drawn.



* The original position of the pen is set as the start point.
The line becomes increasingly rough as it is drawn.

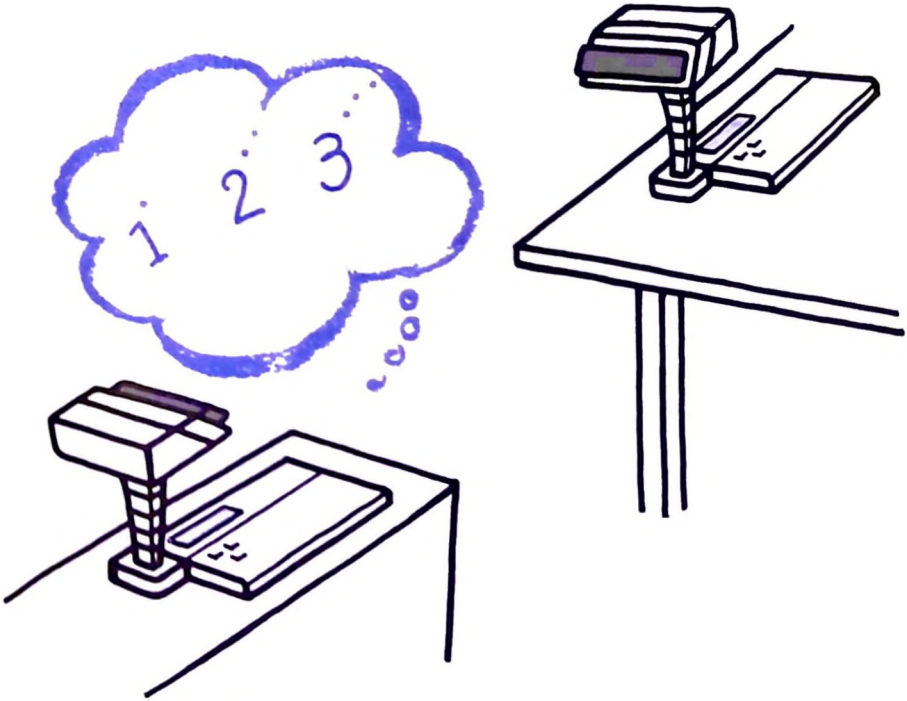


4.5 Optical coupler

One Optical Coupler X-721 can be connected to one X-07, and another Optical Coupler X-721 can be connected to another X-07 for cableless communication.
In this section, programs for transmitting/receiving data between two X-07s using optical couplers are introduced.

A. Transmitting numeric values

Let's write programs for transmitting numeric values from one X-07 (A) to another (B), using optical couplers.



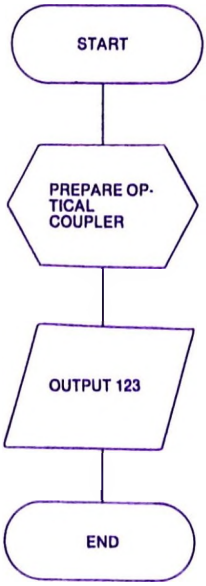
To transmit numeric value 123 to B from A, the programs below must be created in each X-07.

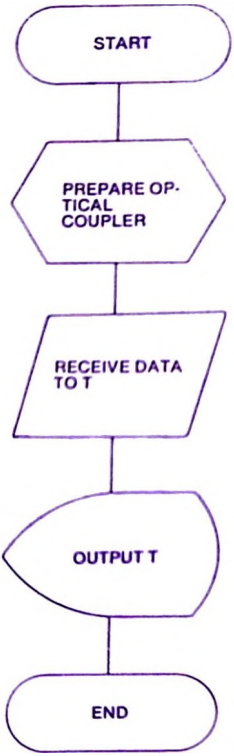
A

```
10 REM OPT 1A
20 INIT #1,"OPT:"
30 PRINT #1,123
40 END
```

B

```
10 REM OPT 1B
20 INIT #1,"OPT:"
30 INPUT #1,T
40 PRINT T
50 END
```





Let's first look at program A.

20 INIT #1, "OPT:"

This line readys the optical coupler for operation.

30 PRINT #1, 123

Line 30 transmits numeric value 123 through the optical coupler. In program B, the other optical coupler is also made ready for operation at Line 20.

30 INPUT #1, T

At Line 30, the datum received through the optical coupler is stored in the variable T. And, at line 40, the contents of T are displayed.

Now, let's execute the programs.

- First, when program B is executed, at the INPUT statement of Line 30, it will wait to receive data.
- Next, when program A is executed, datum 123 is transmitted through the optical coupler.
- B receives this datum, goes out of the waiting mode, and executes Line 40 to display 123 on the LCD.

Please note that if program A is executed before program B, B will not wait for data reception, and 123 sent from A will not be received.

B. Transmitting characters

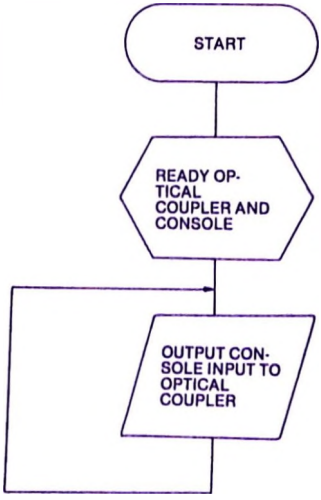
Now, let's write programs which display on B, characters typed at A.

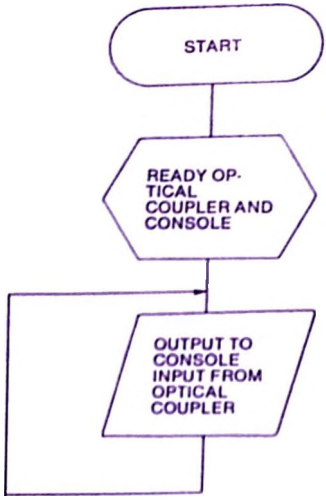
A

```
10 REM OPT 2A
20 INIT #1,"OPT:"
30 INIT #2,"CON:"
40 OUT #1,INP(#2)
50 GOTO 40
```

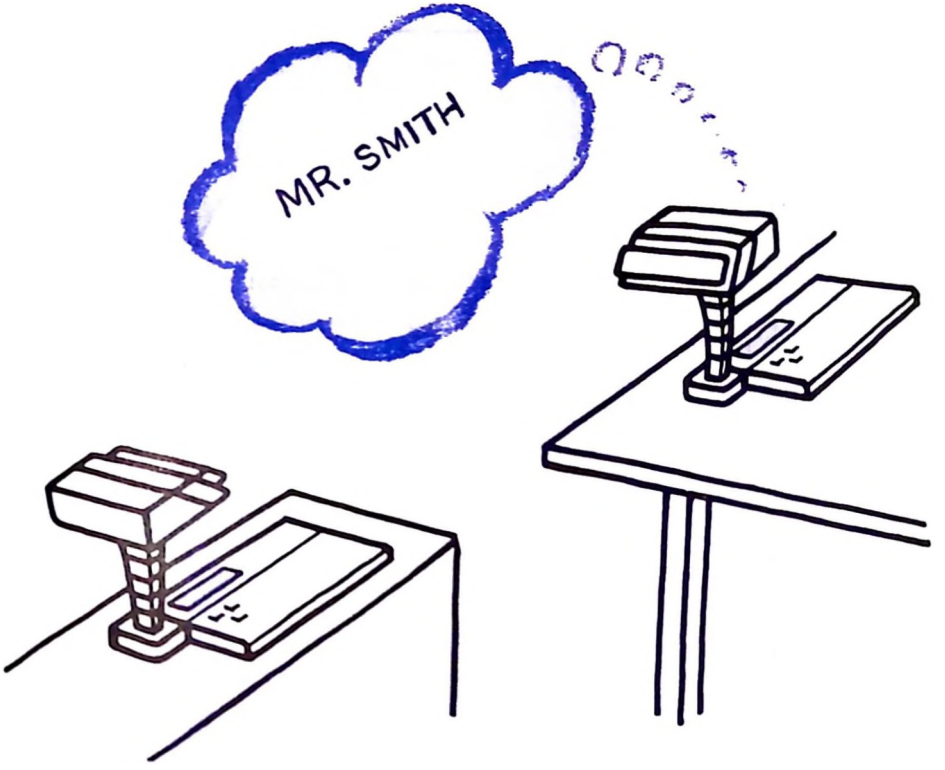
B

```
10 REM OPT 2B
20 INIT #1,"OPT:"
30 INIT #2,"CON:"
40 OUT #2,INP(#1)
50 GOTO 40
```





As before, please execute program B before executing program A. When program A is executed, characters typed at A are not displayed on the LCD of A, but on the LCD of B.



New Instructions

OUT #	Outputs to # <file number>.
INP #	Reads value of # <file number>.

C. Transmitting a file

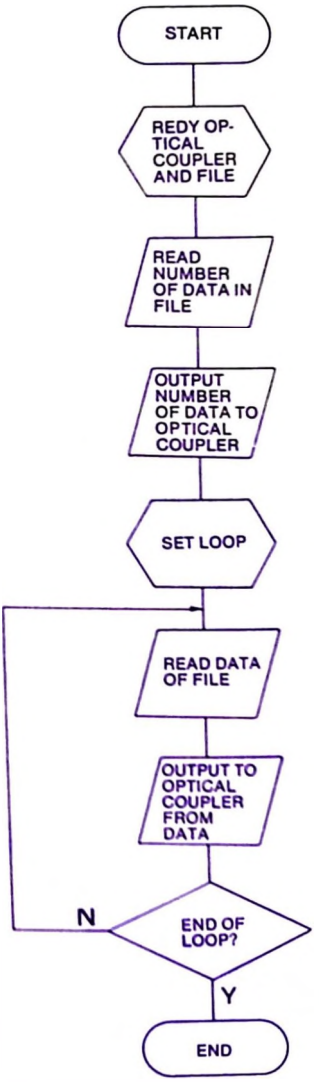
The next programs copy the TEL RAM file of A into RAM of B.

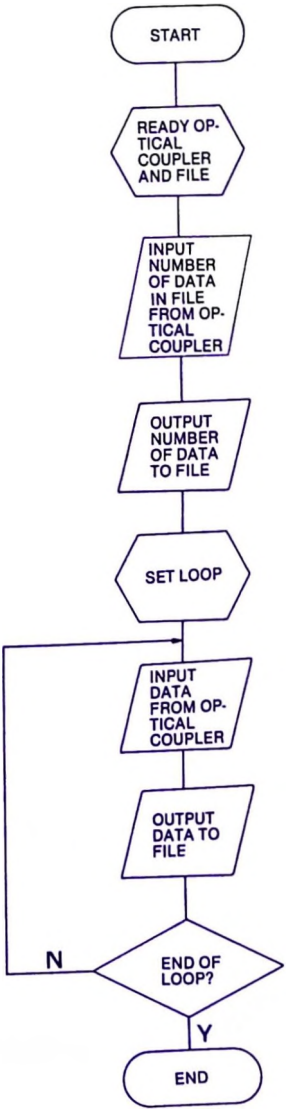
A

```
10 REM OPT 3A
20 INIT #1,"OPT:"
30 INIT #2,"TEL"
40 INPUT #2,N
50 PRINT #1,N
60 FOR I=1 TO N
70 INPUT #2,N$,T$   Reads TEL.
80 PRINT #1,N$,T$   Writes into OPT.
90 NEXT I
100 END
```

B

```
10 REM OPT 3B
20 INIT #1,"OPT:"
30 INIT #2,"TEL",100,"D"
40 INPUT #1,N
50 PRINT #2,N
60 FOR I=1 TO N
70 INPUT #1,N$,T$   Reads from OPT.
80 PRINT N$,T$
90 PRINT #2,N$,T$   Writes into TEL.
100 NEXT I
110 END
```


START
REDY OP-TICAL COUPLER AND FILE
READ NUMBER OF DATA IN FILE
OUTPUT NUMBER OF DATA TO OPTICAL COUPLER
SET LOOP
READ DATA OF FILE
OUTPUT TO OPTICAL COUPLER FROM DATA
N
END OF LOOP?
Y
END


START
READY OP-TICAL COUPLER AND FILE
INPUT NUMBER OF DATA IN FILE FROM OP-TICAL COUPLER
OUTPUT NUMBER OF DATA TO FILE
SET LOOP
INPUT DATA FROM OP-TICAL COUPLER
OUTPUT DATA TO FILE
N
END OF LOOP?
Y
END

The TEL file is the telephone directory file which we created in section 3.5. At A, TEL file is read, and then transmitted through the OPT. At B, data are read through the OPT, and then written into the TEL file.
When these programs are executed, a TEL RAM file, exactly same as the one in A, is created in B.

File

Some of you might have difficulties with technical words when reading a computer manual. Of these technical words, the word "file" might be explained as "collection of informations, etc.". However, some of you might not be completely satisfied with this explanation.
Usually, the word "file" gives a image of a bundle of papers in a binder, etc. This image is actually all right, but a computer does not have skillful fingers and eyes to turn and read the papers. Therefore, with a computer, the bundle of papers is put in a different form, and everything is called a "file".
For example, in the X-07, programs and data are written into or read from an internal memory called a cassette or RAM file. And, all the programs and data arranged together in some place are collectively called a "file".

D. Conversation

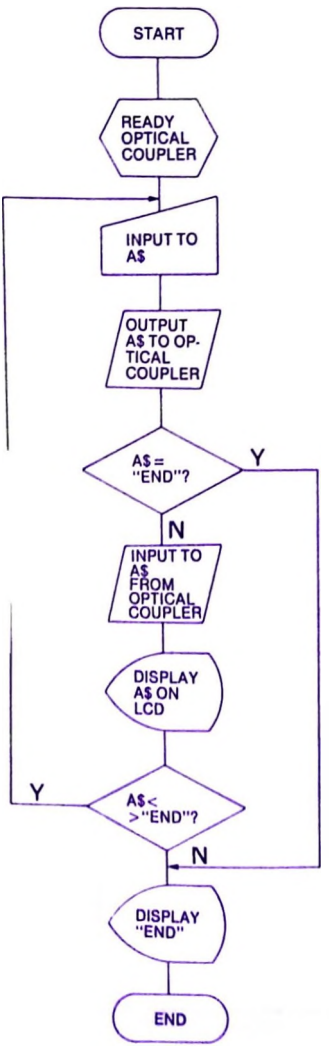
This time, let's write programs for a conversation between A and B.

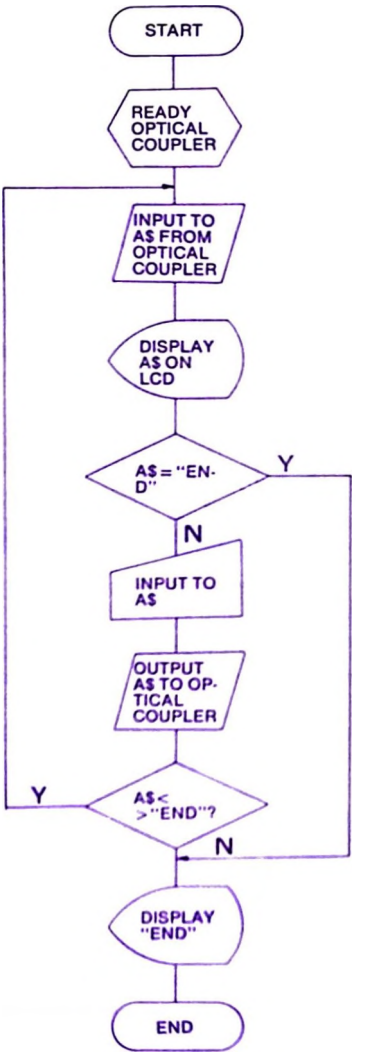
A

```
10 REM OPT 4A
20 INIT#1,"OPT:"
30 INPUT A$
40 PRINT#1,A$
50 IF A$="END" THEN 90
60 INPUT#1,A$
70 PRINT A$
80 IF A$<>"END" THEN 30
90 PRINT "END"
100 END
```

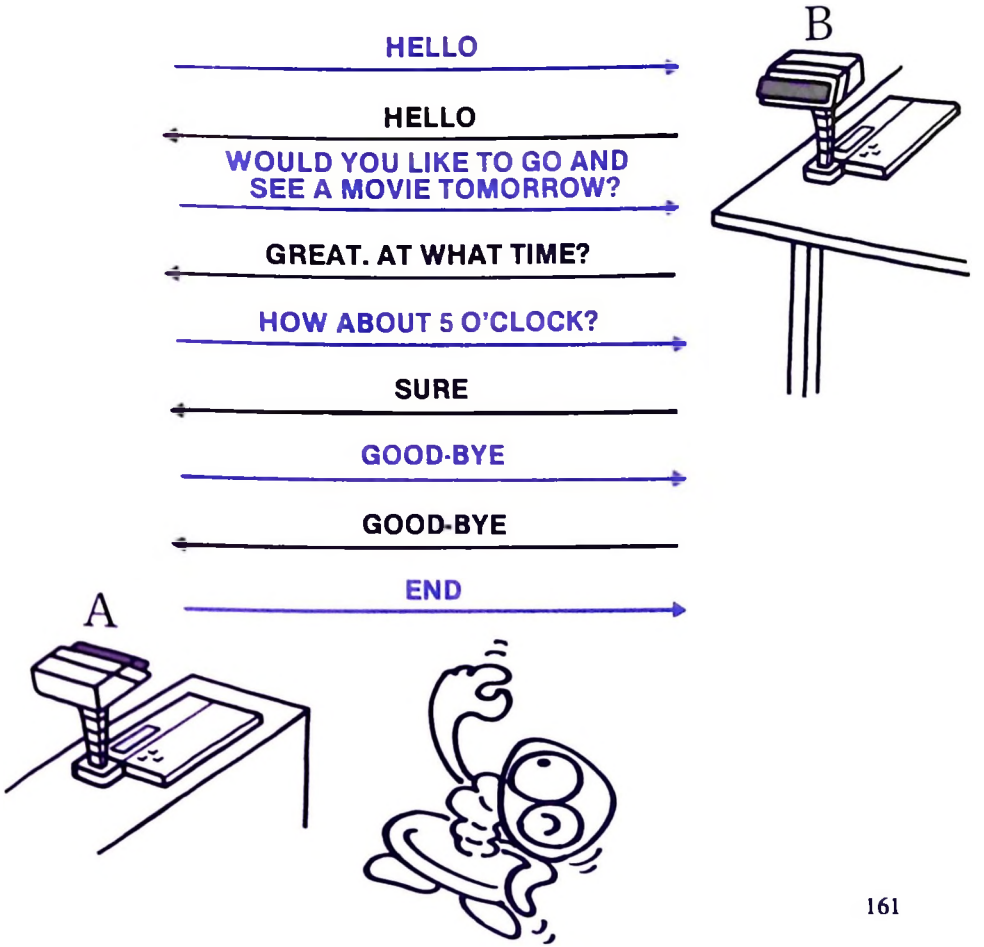
B

```
10 REM OPT 4B
20 INIT#1,"OPT:"
30 INPUT#1,A$
40 PRINT A$
50 IF A$="END" THEN 90
60 INPUT A$
70 PRINT#1,A$
80 IF A$<>"END" THEN 30
90 PRINT "END"
100 END
```

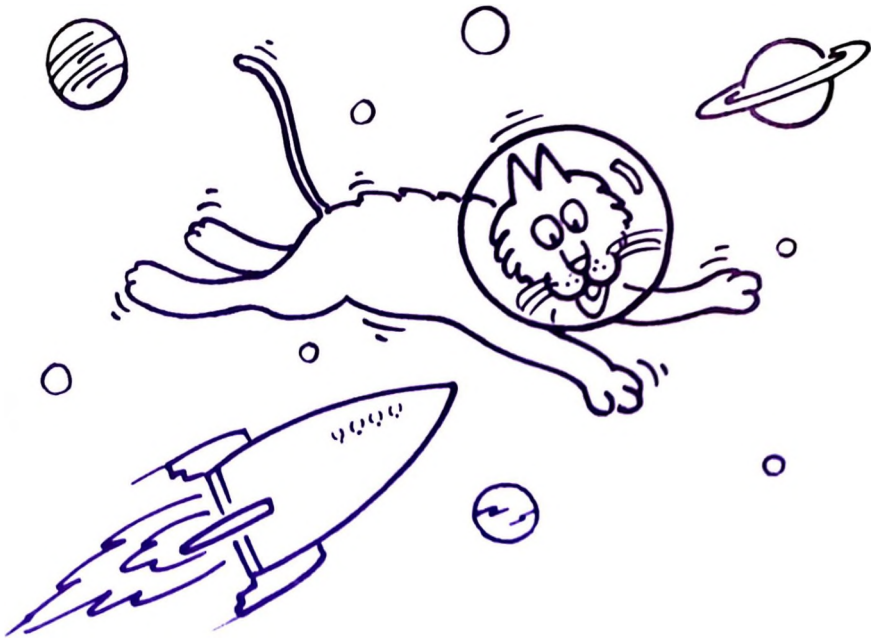

START
READY OPTICAL COUPLER
INPUT TO A$
OUTPUT A$ TO OPTICAL COUPLER
A$ = "END"?
Y
N
INPUT TO A$ FROM OPTICAL COUPLER
DISPLAY A$ ON LCD
A$< >"END"?
Y
N
DISPLAY "END"
END


START
READY OPTICAL COUPLER
INPUT TO A$ FROM OPTICAL COUPLER
DISPLAY A$ ON LCD
A$ = "END"?
Y
N
INPUT TO A$
OUTPUT A$ TO OPTICAL COUPLER
A$< >"END"?
Y
N
DISPLAY "END"
END

First , A types a message. When the RETURN key is depressed, the message is transmitted to and displayed on B.
Next, B looks at this message and transmits a message to A. In this way, messages an be communicated between A and B. To terminate the programs, transmit "END" from A or B.



The final program is the space trader game. The summary, the explanation and the actual execution of this program are given in sections. The program list is at the end of "5.2 Explanation of the program".

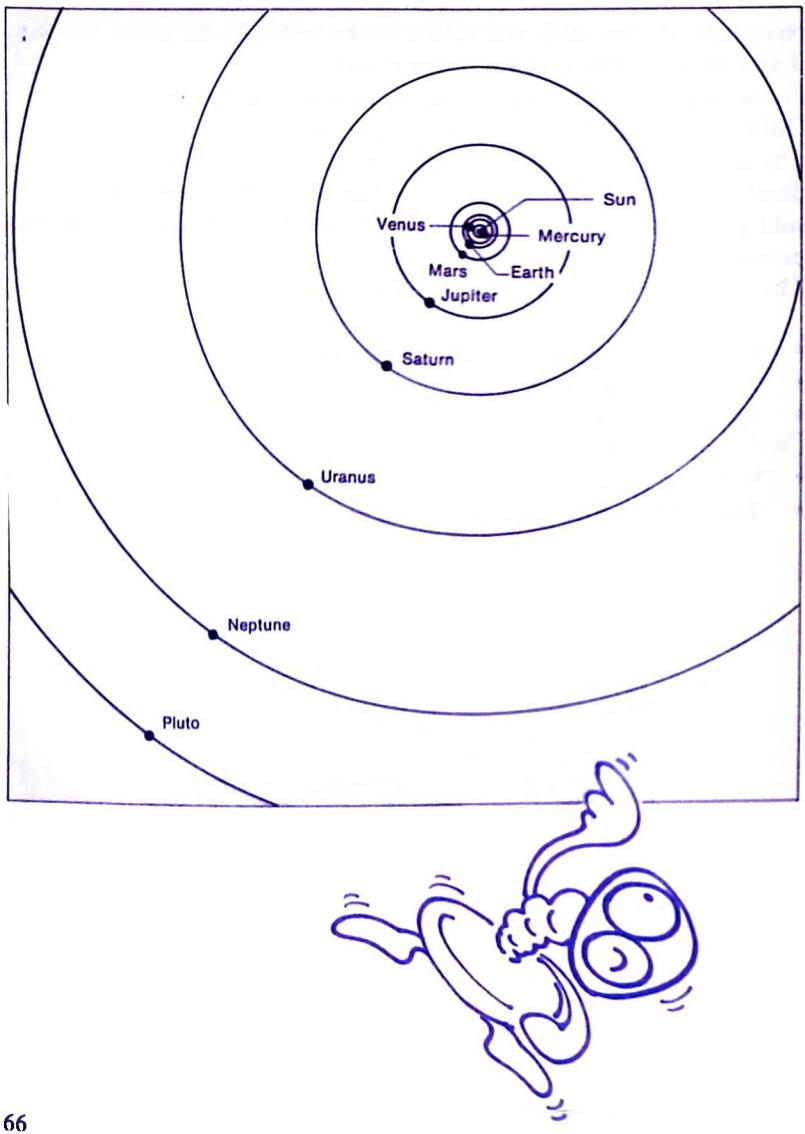


5.1 Summary of the game

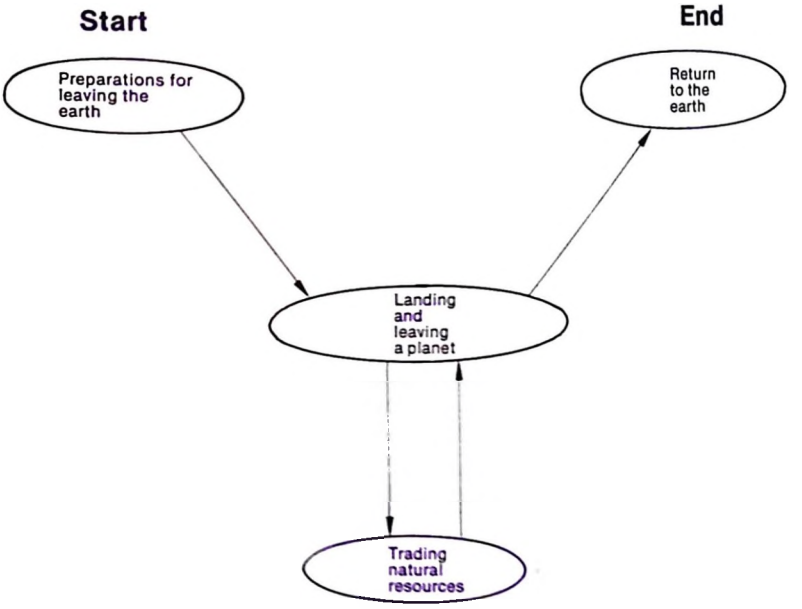
You leave the earth seeking tremendous wealth. You shuttle from one planet to another in the solar system, trading various natural resources. At the end, you return to the earth. At this point, you win if you have profit, and you loose if not.
Following natural resources can be traded at each planet:
Gold, carbon, iron, hydrogen, oxygen, and uranium
The prices of these natural resources differ at each planet. Accordingly, the basic strategy is to buy them at planets where they are sold at low prices, and sell them at planets where they are sold at high prices.
The basic rules of this game are as follows:

- You leave from and return to the earth.
- You cannot go to a same planet twice.
- As a rule, you must visit all the planets. However, you can return to the earth between visits, but you will be fined.
- The game is over when you run out of energy.
- The game is also over when you get too close to the sun.

You can visit the following planets (In order of closeness to the sun).



This game proceeds through the following events:



☐Preparations for leaving the earth

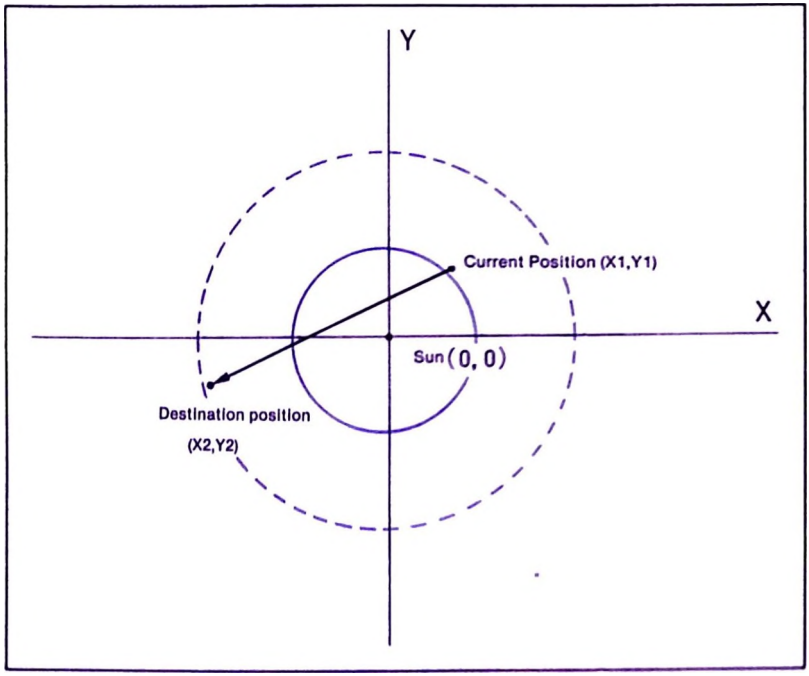
When leaving the earth, you are given the following amounts of money, energy, and natural resources.

Money: 4700 Energy: 50000
Gold: 10 Carbon: 30 Iron: 20 Hydrogen: 40
Oxygen: 40 Uranium: 20

You can also receive only money and use it to buy all the natural resources. In such a case, the amount of money you will receive is 10000.

☐ Arriving at and leaving from a planet

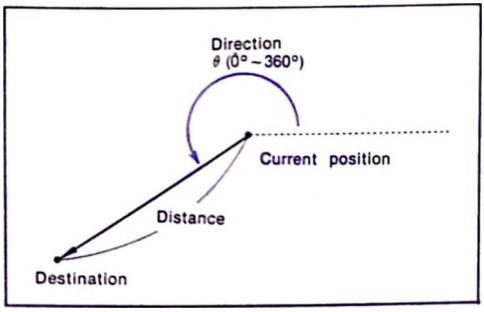
First, specify the planet you want to visit. When it is specified, its coordinates are returned. These coordinates give the position of the planet in a two-dimensional plane with the sun as its centre. The coordinates are shown in 10000 km units, and all the inputs/outputs of this game are done in this unit.



At departure time, direction and distance must be given. Direction and distance can be calculated from the current position and the destination position. When distance is less than 100 million km, direction and distance can be calculated by the computer. When it is more

than 100 million km, calculate them by hand, and move the spaceship within 100 million km of the destination position, then let the computer calculate the remaining direction and distance. Also, energy consumption is proportional to the square of distance travelled. Therfore, when travelling over 100 million km, it is very dangerous unless you warp in serveral stages.

When the spaceship comes near a planet, you must prepare for landing. At this stage, the planet and the spaceship are displayed on the LCD. Use the cursor keys and move the spaceship in the direction of the planet. Landing is successful when the spaceship passes over the planet.



☐Trading natural resources

When you land on the planet, you must first sell your natural resources, and then you must buy the natural resources of the planet. The prices differ at each planet. They also fluctuate every day. However, there is a certain market standard.

When selling the natural resources, if you set the prices too high, you will not be able to sell too great a quantity. And, if you set them too low, you will not make much of a profit. You can show your ability by selling your natural resources at prices which will give you the maximum profit.

When buying natural resources, if you buy in large quantities, the prices will be low. And, if you buy too little, the prices will be high.

When trading is over, you must leave for another planet. When you want to terminate the program, return to the earth.

□Returning to the earth

When you return to the earth, your account is settled as follows:

- All your natural resources and the remaining energy are converted into money at 90% of the current market value.
- You are given a bonus if you return within 180 days.
- You are fined if you did not visit a planet or planets.

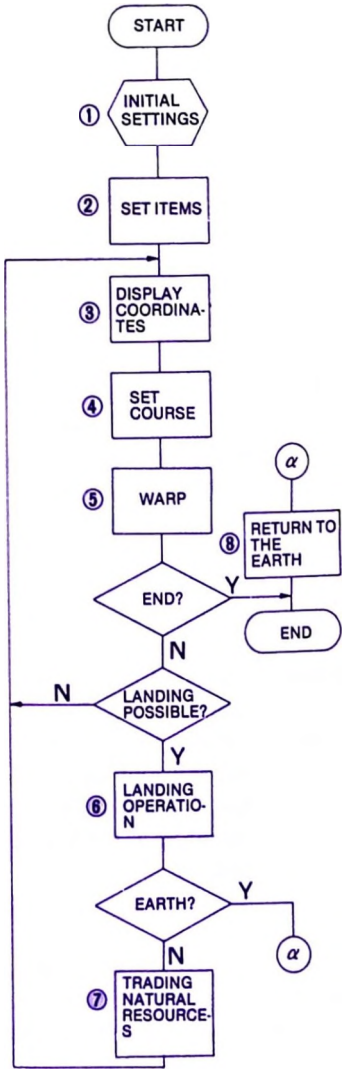
At the end, the game is terminated after displaying the total amount of money, the time you spend on your trip, and your profit.

5.2 Explanation of the program

As shown on the flowchart, the program consists of a main routine and several subroutines.

A. Main routine

The program control flows in the following order:
First, ① the initital market prices of natural resources, etc. are set. ② Money, energy and natural resources that you are taking along on the space trip are procured.
When the preparation for departure is over, ③ the destination planet is input, and the coordinates of the planet are returned. ④ Then, the course (direction and distance) of the spaceship is set.
When the course is set, ⑤ the spaceship is moved according to the set course. If energy runs out during the flight, the program is terminated.
If the spaceship is within the landing range of the planet, ⑥ landing operation stage is set.
If the spaceship is not within the landing range of the planet, the control is returned to ③.



When landing is successful, ⑦ trading of natural resources is conducted.
When trading is over, the control is returned to ③ for a flight to another planet.
When the spaceship returns to the earth, ⑨ the account is settled, the results are displayed, and the game is terminated.

① Initial settings

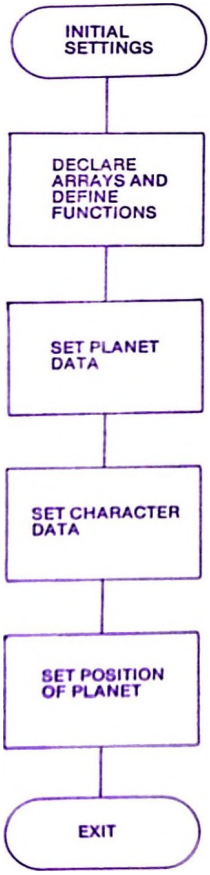
Several initital settings are completed at this stage. First, the following four functions are defined:

FNA : Seeks the angle, in degrees, of the line which connects the (A, B) coordinates and the (C, D) coordinates.

FNK : Seeks the distance between the (A,B) coordinates and the (C,D) coordinates.

FNR : Changes the angle to radians.

FNS : Returns the minimum price for buying B amount of natural resources at the market price A.



At Lines 80 to 200, the following data are set, and read into the array variable P:

Data set in P (I, J)

I \ J	0	1	2	3	4	5	6	7	8	9	10	
0	0	0	0	0	0	0	0	0	0	0	0	You
1	57	.24	1	0	0	90	40	50	30	40	40	Mercury
2	108	.62	2	0	0	100	20	40	10	10	50	Venus
3	150	1	2	0	0	100	30	40	20	10	70	Earth
4	228	1.88	1	0	0	100	30	30	20	20	60	Mars
5	778	11.86	28	0	0	110	10	50	10	20	60	Jupiter
6	1426	29.46	24	0	0	90	10	50	10	20	70	Saturn
7	2871	84.08	10	0	0	90	40	40	10	30	40	Uranus
8	4497	164.82	10	0	0	80	30	40	10	30	80	Neptune
9	5914	248.54	8	0	0	90	40	30	20	40	60	Pluto

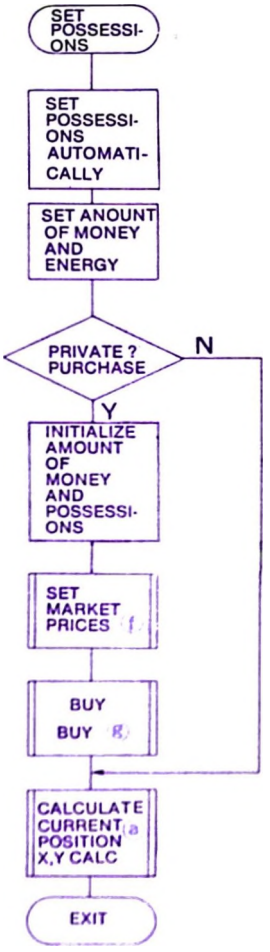
Distance from the sun (Million km)	Revolution time (Year)	Radius (25 million km)	Initial position (Radians)	Visiting flag	Gold	Carbon	Iron	Hydrogen	Oxygen	Uranium
					Market prices					

Lines 210 to 250 set character string data, and read them into the array variable A$.
Lines 260 to 290 set the current position of a planet into the array P(I,3). The current posituion is shown in radians, and is generated by using random numbers. Line 290 sets your current position as the position of the earth.

② Setting your possessions

At Lines 300 to 420, initial setting of your possessions, and the date of departure is set. At Lines 300 to 330, the amount of natural resources, money and energy are set.

Next, you are asked whether private purchases are planned. If so, natural resources are bought with M = 10000 amount of money, and P(0, I) is reset. Finally, 1 is added to D, the number of days passed, and the coordinates of the current position (earth) are obtained.



Money M = 4700
Energy E = 50000

0	1	2	3	4	5	6	7	8	9	10
					10	30	20	40	40	20

P (I, J) Natural resources

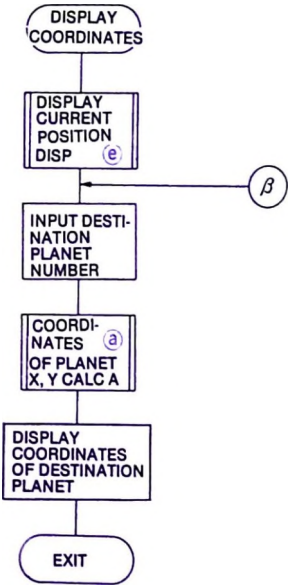
Gold Carbon Iron Hydrogen Oxygen Uranium

③ Displaying coordinates

Lines 430 to 490 calculate and display the coordinates of the current position and the coordinates of the destination planet. The destination planet can be specified with the follow numbers:

1 = Mercury 2 = Venus
3 = Earth 4 = Mars
5 = Jupiter 6 = Saturn
7 = Uranus 8 = Neptune
9 = Pluto

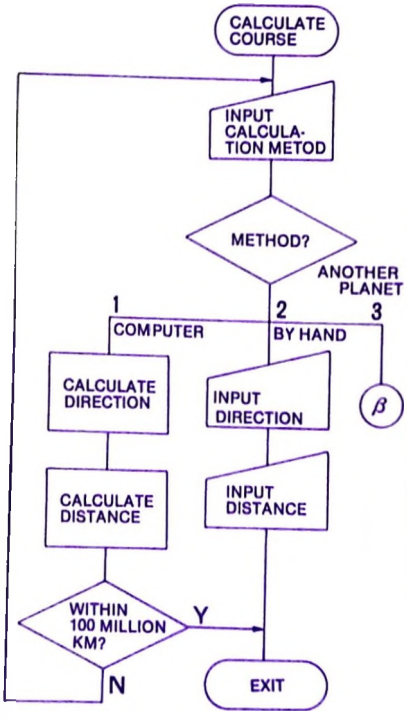
Another day passes during this stage.



④ Computing the course

Lines 500 to 690 calculates the course of the spaceship. The method of calculation is input at Line 510. The method of calculation differ according to the number input.

1. **The computer automatically calculates the direction and the distance.**
2. **The direction and the distance are typed on the keyboard.**
3. **Checks the coordinates of another planet.**

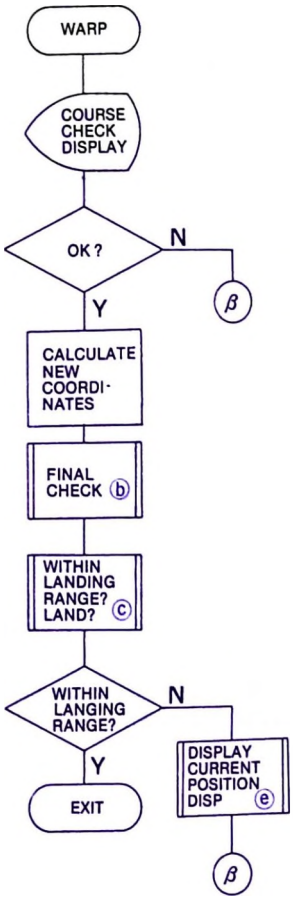


Selection is decided at Lines 510 and 520. When N = 1, the control is transferred to Line 540, to Line 660 when N = 2, to Line 450 when N = 3, and back to Line 510 when N is any other number. The computer calculates only when the distance is within 100 million km. Accordingly, when the distance comes out to more than 100 million km, the program displays that the distance must be calculated by hand, and returns to the line for input of calculation method.
When calculation method 3 is input, the coordinate display is shown again.

⑤ Warp

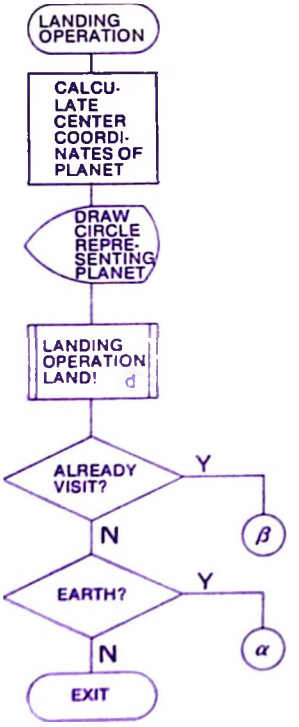
At Lines 700 to 810, the current position of the spaceship is moved according to the calculated course (direction and distance).
First, the course check display is shown. If it is wrong, input any key other than "Y", to go back to the coordinate display.

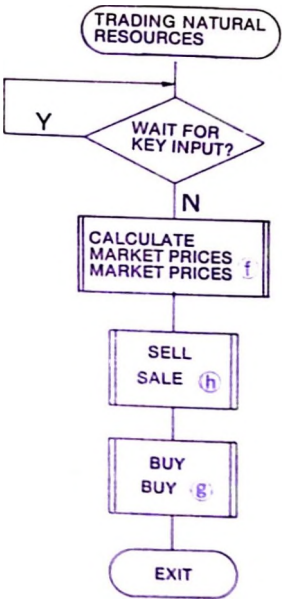
If it is all right, energy is consumed (Line 770), and a new current position is set (Lines 780 and 790).
Also, this stage checks whether there is energy left, and whether you did not crash into the sun (GOSUB 1060).
Finally, this stage checks whether the current position is within the landing range. If not, the coordinate display with the current position is shown.



⑧ Landing

Landing operation is executed at Lines 820 to 890.
First, the LCD is cleared (Line 820). Next, the centre coordinates of the planet is calculated (Line 840), and a circle representing the planet is displayed (Line 850). The spaceship is represented by a dot. The spaceship is moved with the cursor keys. When the spaceship passes over the planet, landing operation is finished (GOSUB 1210). If the planet has already been visited, warning sound is emitted, and the corrdinate display is shown again (Line 880). Or, if the planet on which you have landed is the earth, the control is transferred to the "upon returning to the earth" stage (Ling 890).





⑦ Trading natural resources

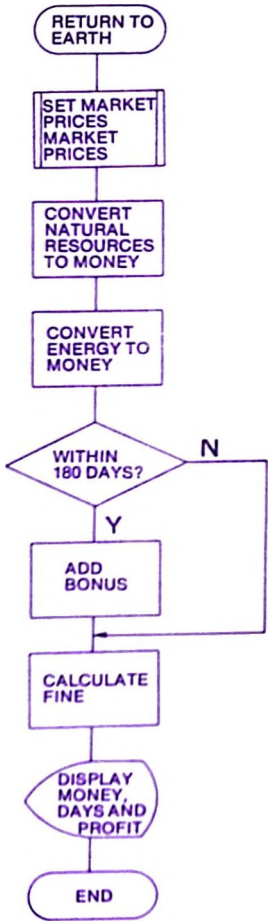
Natural resources are traded at Lines 900 to 940.
When a key is depressed (it can be any key), the control is transferred in the following order:

Setting the market prices (GOSUB 1510)
Selling the natural resources (GOSUB 1810)
Buying the natural resources (GOSUB 1610)
When trading is over, you leave for the next planet (Line 940).

⑧ Upon returning to the earth

Lines 2000 to 2210 execute the final process upon returning to the earth.

First, the market prices are set (Line 2030). Next, the natural resources you have brought home are converted into money accordig to the market prices (Lines 2050 to 2070). The remaining energy is also converted into money (Line 2100). A bonus is added if you returned within 180 days (Line 2110). A fine is subtracted if you did not visit a planet or planets (Lines 2120 to 2160). Finally, the total amount of money, the days spent on the trip, and the profit are displayed, and the game is terminated.



B. Subroutines

This program uses the following 8 subroutines:

- ⓐ **Calculation of the coordinates of a planet**
- ⓑ **Check**
- ⓒ **Check of landing range**
- ⓓ **Landing operation**
- ⓔ **Display of current position, etc.**
- ⓕ **Set market prices**
- ⓖ **Buy natural resources**
- ⓗ **Sell natural resources**

ⓐ Calculation of the coordinates of a planet (Lines 1000 to 1040)

Input:	P:	planet number (0 ~ 9)
	D:	current date
Output:	X:	X-coordinate of current position
	Y:	Y-coordinate of current position

This subroutine calculates and returns the current position (X,Y) of the input planet.

R (angle) = 6.28/revolution cycle (year)/365 × days + initial angle (Line 1010)
X (10000 km) = cos(R) × distance from the sun × 100 (Line 1020)
Y (10000 km) = sin(R) × distance from the sun × 100 (Line 1030)

ⓑ Check (Lines 1050 to 1080)

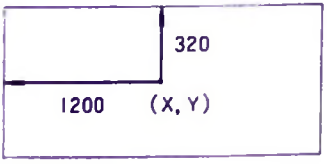
Input:	E	(energy)
	X1, Y1	coordinates of current position
Output:		Conditions for termination

When the amount of remaining energy is negative (Line 1060), or the current position is too close to the sun (Line 1070), a message is displayed, and the program is terminated. Although it is ordinarily better to avoid leaving a subroutine by a GOTO sentence, there is no problem here because the program is terminated immediately at Line 2210.

c Check of landing range (Line 1100 to 1160)

Input: X, Y: position of planet
X1, X2: current position
Output: P: planet number
≦9: landing operation possible
>9: landing operation not possible

Landing operation is possible when the current position (X1, Y1) is at the shown range.



Line 1120 checks the position of each planet, and calculates their distances from the current position. If landing is possible, the loop is terminated by setting I to 20. If you are too far from any planet, the loop is terminated anyway, and I is set to 10. This is judged in Line 1150, and P is set to 10 when landing is possible, and the control is returned to the main routine. This is a somewhat complicated procedure, but it is used because one should avoid leaving a FOR ~ NEXT loop by a GOTO statement.

(d) Landing operation (Lines 1200 to 1320)

Input: XD, YD: centre coordinates of planet
P: planet number (1 ~ 9)
I$: input from the keyboard.
Output: Display until you land.

First, the initital position of the spaceship is generated using random numbers (Line 1210). When the initial position is within the planet, the initial position is regenerated (Line 1220).
The spaceship is moved and its speed is changed with the cursor keys.

→ CHR$ (28) →PX = PX + 1

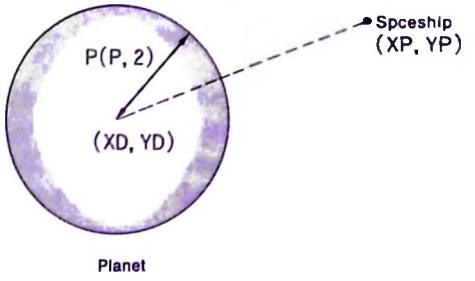
← CHR$ (29) →PX = PX − 1

↑ CHR$ (30) →PY = PY − 1

↓ CHR$ (31) →PY = PY + 1

The position of the spaceship is indicated by XP and YP. PX and PY indicate the speed of the spaceship, and they vary according to whether the result of the logical expressions in Lines 1240 and 1250 are true (− 1) or false (0). They are used at Line 1280 to move the current position of the spceship. In other words, once a cursor key is depressed, the spaceship is moved at a fixed speed.
The subroutine also checks to prevent the spaceship from moving out of the LCD (Lines 1290 to 1300).

The spceship is safely landed when it passes over the planet.



(e) Displays (Lines 1400 to 1440)

Input: M: money
E: energy
X1, Y1: coordinates of current position
Output: Display on LCD
Displays the current amount of money and energy you have, and the coordinates of your current position.

(f) Setting the market prices (Lines 1500 to 1530)

Input: P: planet number
P (P,I): current market prices
Output: P (P,I): new market prices

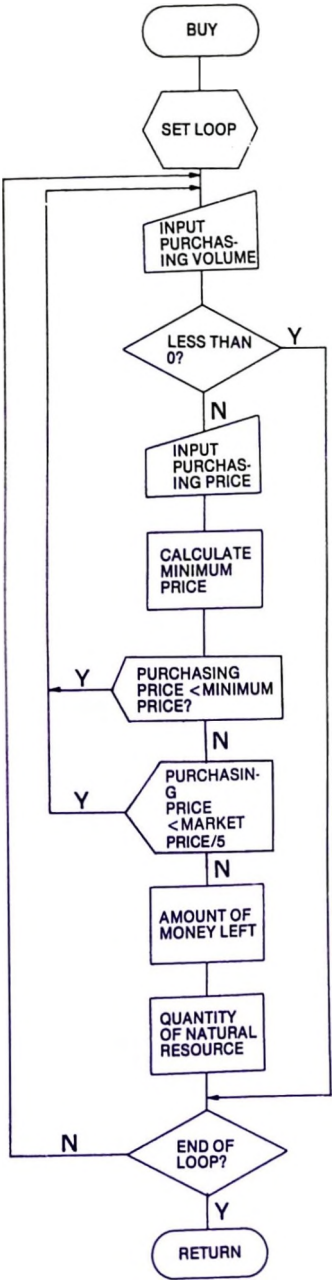
A new market price is set as follows:
new market price = current market price $\times$ (RND(1) $\times$ .4 + 0.8)

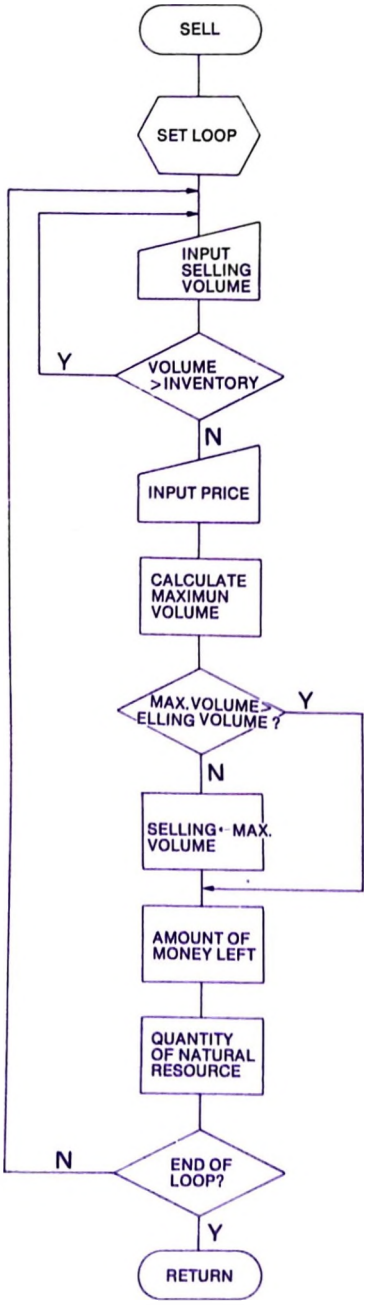
(g) Buying natural sources (Lines 1600 to 1730)

Input: M: money
P(0,I): your natural resources

Output: M: money
P(0,I): your natural resources
D: passing of time

The purchasing volume and price of each natural resource are input (Line 1650 to 1670). The anount of money is decreased, and the amounts of natural resources are increased accordingly (Line 1710). However, as the minimum price at each purchasing volume is already set (Line 1680), the subroutine requests to reinput the price if is below this minimum price. Reinput is also necessary if the input price is less than 1/5 of the market price (Line 1690). Date is advanced by one day with purchasing of each natural resource. A reinput adds an extra day.





h Selling natural resources (Lines 1800 to 1920)

Input M: money
P(0,I): your natural resources
Output: M: money
P(0,I): your natural resources

The selling volume and price for each natural resource are input (Lines 1820 to 1860). And, the amount of money is increased and the amount of each natural resource is decreased accordingly. However, the maximum volume of sale for each natural resource is determined from the input price and the market price.
Maximum volume of sale = (market price/price)^{10} (Line 1870)
Therefore, it is not always possible to sell the desired input volume (Lines 1880 to 1900).
One day is spent on sale of each natural resource.

© Game List

```
① 10 REM *** SPACE TRADE ***
  20 DEFINT I,N:R=RND(0)
  30 DIM P(9,10),A$(16)
  40 DEFFNA(A,B,C,D)=ATN((D-B)/(C-A))*180/
  3.1415926535898O6#
  50 DEFFNK(A,B,C,D)=SQR((C-A)^2+(D-B)^2)
  60 DEFFNR(A)=A/180*3.141592653589806#
  70 DEFFNS(A,B)=A*(.7-.4*(ATN(B/40-1)))
  80 FOR I=1 TO 9
  90 FOR J=0 TO 10
  100 READ P(I,J)
  110 NEXT J,I
  120 DATA 57,.24,1,0,0,90,40,50,30,40,40
  130 DATA 108,.62,2,0,0,100,20,40,10,10,5
  0
  140 DATA 150,1.2,0,0,100,30,40,20,10,70
  150 DATA 228,1.88,1,0,0,100,30,30,20,20,
  60
  160 DATA 778,11.86,28,0,0,110,10,50,10,2
  0,60
  170 DATA 1426,29.46,24,0,0,90,10,50,10,2
  0,70
  180 DATA 2871,84.08,10,0,0,90,40,40,10,3
  0,40
  190 DATA 4497,164.82,10,0,0,80,30,40,10,
  30,80
  200 DATA 5914,248.54,8,0,0,90,40,30,20,4
  0,60
```

```
210 FOR I=0 TO 16
220 READ A$(I):NEXT
230 DATA "YOUR ","MERCURY","VENUS","EART
H","MARS"
240 DATA "JUPITER","SATURN","URANUS","NE
PTUNE","PLUTO"
250 DATA "SPACE","GOLD","CARBON","IRON",
"HYDROGEN","OXYGEN","URANIUM"
260 FOR I=1 TO 9
270 P(I,3)=RND(1)*6.2831853071796O3#
280 NEXT
290 P(0,3)=P(3,3)
300 FOR I=5 TO 10
310 READ P(0,I):NEXT
320 DATA 10,30,20,40,40,20
330 M=4700:E=50000!
340 REM *** START ***
350 CLS:P=3
360 PRINT "PRIVATE PURCHASE?"
370 Y$=INKEY$:IF Y$="" THEN 370
380 IF Y$<>"Y" THEN 410
390 FOR I=5 TO 10:P(0,I)=0:NEXT
400 M=10000:GOSUB 1510:GOSUB 1610
410 D=D+1:GOSUB 1010
420 X1=X:Y1=Y
430 REM *** SPACE ***
440 CLS:GOSUB 1410
450 PRINT "PLANET(1-9)?";
460 N=INSTR("0123456789",INKEY$)-1
470 IF N<=0 THEN 460
480 PRINT:P=N:D=D+1:GOSUB 1010
```

```
490 PRINT A$(N);"'S COORDINATES ARE",INT
(X);" ";INT(Y)
500 PRINT"1=COM:2=MAN:3=ANOTHER?";
510 ON VAL(INKEY$) GOTO 540,660,450
520 GOTO 510
530 REM *** COMP ***
540 PRINT"   COMPUTING COURSE TO ";A$(P)
550 FOR I=1 TO 24
560 BEEP I,1:NEXT
570 IF X1>X THEN 610
580 IF Y1>Y THEN 600
590 C1=FNA(X1,Y1,X,Y):GOTO 620
600 C1=FNA(X1,Y1,X,Y)+360:GOTO 620
610 C1=FNA(X1,Y1,X,Y)+180
620 K=FNK(X1,Y1,X,Y):R=RND(1)^6
630 IF K<10000 THEN 690
640 BEEP 4,10:PRINT "TOO FAR. MAN CAL.!"
:GOTO 500
650 REM *** MANUAL ***
660 PRINT "   MANUAL NAVIGATION"
670 INPUT "DIRECTION(0-360)";C1
680 INPUT "DISTANCE=";K:R=0
690 C=FNR(C1)

700 REM *** WARP ***
710 PRINT "DIRECTION=";C1
720 PRINT "DISTANCE=";K
730 INPUT "OK";Y$
740 IF Y$<>"Y" THEN 450
750 PRINT "WARP!"
760 BEEP 10,20
770 E=E-100-K^2/10000000#
```

```
780 X1=X1+COS(C)*K*(RND(1)*R+1)
790 Y1=Y1+SIN(C)*K*(RND(1)*R+1)
800 GOSUB 1060:GOSUB 1110
810 IF P=10 THEN GOSUB 1410:GOTO 450
820 PX=0:PY=0:CLS
830 GOSUB 1010
840 XD=60+(X-X1)/20:YD=16-(Y-Y1)/20
850 CIRCLE(XD,YD),P(P,2)
860 GOSUB 1210:BEEP 15,10:CLS
870 PRINT "LANDED ON ";A$(P)
880 IF P(P,4)=1 THEN PRINT "BEEN HERE BE
FORE!":BEEP 2,10:GOTO 440
890 P(P,4)=1:IF P=3 THEN 2010
900 IF INKEY$="" THEN 900
910 PRINT "SELL":GOSUB 1510:GOSUB 1810
920 PRINT "BUY":GOSUB 1610
930 PRINT "DEPART!":BEEP 5,5
940 GOTO 440
1000 REM *** X,Y CALC ***
1010 R=6.283185307179605#/P(P,1)/365.242
*D+P(P,3)
1020 X=COS(R)*P(P,0)*100
1030 Y=SIN(R)*P(P,0)*100
1040 RETURN
1050 REM *** CHECK ***
1060 IF E<0 THEN PRINT "NO ENERGY":GOTO
2210
1070 IF X1^2+Y1^2<20000 THEN PRINT "CRASH
INTO SUN":GOTO 2210
1080 RETURN
```

```
1100 REM *** LAND? ***
1110 FOR I=1 TO 9
1120 P=I:GOSUB 1010
1130 IF ABS(X-X1)<1200 AND ABS(Y-Y1)<320
 THEN I=20
1140 NEXT
1150 IF I<11 THEN P=10
1160 RETURN
1200 REM *** LAND! ***
1210 XP=RND(1)*120:YP=RND(1)*32
1220 IF FNK(XD,YD,XP,YP)<P(P,2) THEN 121
0
1230 PSET(XP,YP):I$=INKEY$
1240 PX=PX-(I$=CHR$(28)):PX=PX+(I$=CHR$(
29))
1250 PY=PY+(I$=CHR$(30)):PY=PY-(I$=CHR$(
31))
1260 E=E+20*(I$>CHR$(27) AND I$<CHR$(32)
)
1270 PRESET(XP,YP)
1280 XP=XP+PX:YP=YP+PY
1290 IF XP<0 THEN XP=0 ELSE IF XP>119 TH
EN XP=119
1300 IF YP<0 THEN YP=0 ELSE IF YP>31 THE
N YP=31
1310 IF FNK(XD,YD,XP,YP)<P(P,2) THEN RET
URN
1320 BEEP 5,1:GOTO 1230
1400 REM *** DISP ***
1410 PRINT "M=";MID$(STR$(M),2);
1420 PRINT " E=";MID$(STR$(INT(E)),2);
1430 PRINT " HERE ";INT(X1);INT(Y1)
1440 PRINT A$(P);".";:RETURN
```

```
(f) 1500 REM *** MARKET PRICES ***
    1510 FOR I=5 TO 10
    1520 P(P,I)=INT(P(P,I)*(RND(1)*.4+.8)):N
    EXT
    1530 RETURN
(g) 1600 REM *** BUY ***
    1610 PRINT A$(0);"MONEY=";M
    1620 FOR I=5 TO 10
    1630 PRINT A$(0)+A$(I+6);"=";P(0,I)
    1640 PRINT A$(I+6);"'S MARKET PRICE IS "
    ;P(P,I)
    1650 INPUT "BUY HOW MUCH";N
    1660 IF N<=0 THEN 1720 ELSE D=D+1
    1670 INPUT "PRICE/UNIT";W
    1680 Z=FNS(P(P,I),N)
    1690 IF Z>W OR W*5<P(P,I) THEN PRINT "CA
    N'T SELL AT THAT PRICE":GOTO 1630
    1700 IF N*W>M THEN PRINT "MONEY=";M;"LEF
    T!":GOTO 1640
    1710 M=M-N*W:P(0,I)=P(0,I)+N
    1720 PRINT "MONEY LEFT=";M
    1730 NEXT:RETURN
(h) 1800 REM *** SALE ***
    1810 FOR I=5 TO 10
    1820 PRINT A$(I+6);"'S MARKET PRICE IS "
    ;P(P,I)
    1830 PRINT "ON HAND=";P(0,I):INPUT "SEL
    L HOW MUCH";N
    1840 IF N>P(0,I) THEN PRINT A$(I+6);P(0,
    I) "LEFT!":GOTO 1830
    1850 IF N<=0 THEN 1910 ELSE D=D+1
    1860 INPUT "PRICE/UNIT";W
```

```
1870 Z=(P(P,I)/W)^10
1880 IF Z<N THEN N=Z
1890 P(0,I)=P(0,I)-N
1900 M=M+N*W:PRINT "AT ";W;" SOLD ";N
1910 NEXT
1920 RETURN
2000 REM *** END OF GAME ***
2010 PRINT "RETURNED TO EARTH"
2020 PRINT "SETTLE ACCOUNTS"
2030 FOR I=5 TO 10:P(P,I)=P(P,I)*.6:NEXT
'GOSUB 1510
2040 FOR I=5 TO 10
2050 PRINT "REMAINING ";A$(I+6);":";P(0,
I)
2060 PRINT "MARKET PRICE ";P(P,I);"* ON
HAND ";P(0,I);"="
2070 PRINT P(P,I)*P(0,I):M=M+P(P,I)*P(0,
I)
2080 IF INKEY$="" THEN 2080
2090 NEXT:PRINT "REMAINING ENERGY:";INT(
E);" IS"
2100 PRINT " CONVERTED TO ";INT(E/100):M
=M+INT(E/100)
2110 IF D<180 THEN PRINT "RETURNED EARLY
. GET BONUS OF ";180-D:M=M+180-D
2120 FOR I=1 TO 9
2130 IF P(I,4)=1 THEN 2160
2140 PRINT "DIDN'T GO TO ";A$(I)
2150 PRINT "FINED ";I^3*10:M=M-I^3*10:BE
EP 2*I,5
2160 NEXT
2170 PRINT "EVERYTHING OVER":BEEP 1,10
2180 PRINT "MONEY=";M
2190 PRINT "DAY=";D:"PROFIT=";M-10000
2200 REM *** GAME OVER ***
2210 PRINT "GAME OVER":END
```

How to clearly state ambiguities

As stated in "Do as I do" in chapter 1, there are no words such as "implication" and "appropriate" in the BASIC language. However, in a game, one would like to say, "Mostly like this." "Mostly" must be expressed clearly. Try using the RND function.

English: "X is either 5 or 6, but it is mostly 6."

BASIC: X = 5 + SGN (INT (RND (1)*10))
Thus, 1 out of 10 times, X = 5. Change the position of "*10", and the probablity will change. When a variable is used, the probability changes accordingly.

Let's draw a pattern.

English: "Draw dots all over the LCD. Drow so that the dots are mostly around the center of the LCD."

```
BASIC: 10 FOR I = 0 TO 6.28 STEP .01
       20 N = RND (1) *RND (1)
       30 PSET (SIN(I)*N*60+ 60, COS (I)*N* 16 + 16)
       40 ENXT
```

Dots are drawn in such a way that it is like looking at a galactic system at an angle. Lines 10, 30 and 40 are one way of drawing circles and ellipses. Study the way in which N of Line 20 is created, and devise other ways and do many other interesting things.

For example, a nubula ring:

20 N = (RND (1) + RND (1) + RND (1))/3

There are many other ways of creating N, and some are used in the game in this chapter. "Mostly", can also be expressed without using the RND function (for example, FNS in the game).

Think of other ways of expressing "mostly". It can be fun.

5.3 Execution

The program first asks you whether you intend a private purchase. If Y is typed, the game starts with the purchase of the natural resources. When purchasing is over, or it you did not conduct a private purchase, the current amount of money, level of energy, and your current position are displayed, and then the program asks which planet you want to go to. When the answer is given, the planet's coordinates are returned.

```
PRIVATE PURCHASE?

M=4700 E=50000 HERE —— Money: Energy
 14982  732 ———————— Current position
EARTH.PLANET(1-9)? — (Depress key 4)
MARS'S COORDINATES ARE
-16932  -15271 ———— Position of Mars
```

Next, the program asks for the method of calculating the course. As the distance seems to be more than 100 million km, the calculation is done manually at first. If 3 is typed, you do not warp, and you seek another planet.

```
1=COM:2=MAN:3=ANOTHER?— (Depress key 3)
PLANET(1-9)? ———————— (Depress key 2)
VENUS'S COORDINATES ARE
```

```
 9030  -5925
1=COM:2=MAN:3=ANOTHER?- (Depress key 2)
MANUAL NAVIGATION
DIRECTION(0-360)? 250— (Input 250)
DISTANCE=? 13000——— (Input 13000)
DIRECTION= 250
DISTANCE= 13000
OK?Y ———————————— (Input Y)
WARP!
```

When Y is typed during the confirmation display, the spaceship warps.
When warp is completed, the new coordinates of the spaceship are displayed on the LCD.

```
M=4700 E=49883 HERE
 10535 -11484 ———— (Near Venus)
SPACE.PLANET(1-9)?—— (Depress key 2)
```

Here again the game asks which planet you want to go, and asks the method of calculating the course. As it seems that the spacepaceship is within 100 million km of the target planet, the course is calculated by the computer.

```
VENUS'S COORDINATES ARE
 9491  -5153
1=COM:2=MAN:3=ANOTHER? (Depress key 1)
COMPUTING COURSE TO
VENUS
DIRCTION= 99.367447235642

DISTANCE= 6416.2197709715

OK?Y
WARP!
```

Once in a while, there will be deviation in the computer calculation to make the game more interesting. There fore, you will not always warp to the landing range of the destination planet in one computer calculation.

```
M=4700 E=49778 HERE
 9423 -4443
SPACE.PLANET(1-9)?
VENUS'S COORDINATES ARE
 9762  -4619
1=COM:2=MAN:3=ANOTHER?
COMPUTING COURSE TO
VENUS
DIRECTION= 332.5070166549

DISTANCE= 382.18037033549
OK?Y
WARP?
```

When the spaceship is within the landing range, the LCD is cleared, and the planet and the spaceship are displayed. Move the spaceship over the planet with the coursor keys. After confirming which plant you have landed on, depress a key.

```
LANDED ON VENUS
```

After landing, trade the natural resources.

```
SELL
GOLD'S MARKET PRICE IS 109
ON HAND= 10 SELL HOW MUCH?
```

```
2
PRICE/UNIT? 106
AT 106 SOLD 1
```

```
URANIUM'S MARKET PRICE IS 51
ON HAND= 20 SELL HOW MUCH?
0
BUY
YOUR MONEY= 4806
YOUR GOLD= 9
GOLD'S MARKET PRICE IS 109
? GOLD'S MARKET PRICE IS 109
GOLD'S MARKET PRICE IS 109
BUY HOW MUCH? 0
MONEY LEFT= 4806
YOUR CARBON= 30
CARBON'S MARKET PRICE IS 20
BUY HOW MUCH? 10
PRICE/UNIT? 19
CAN'T SELL AT THAT PRICE
YOUR CARBON= 30
CARBON'S MARKET PRICE IS 20
BUY HOW MUCH? 20
PRICE/UNIT? 19
MONEY LEFT= 4426
```

```
MONEY LEFT=3246
DEPART!
```

When you finish trading, specify and leave for the next planet.
When you finaly return to the earth, the accounts are settled, and the game is over.

```
M=1110 E=22143 HERE
-4402  21074
SPACE.PLANET(1-9)?
EARTH'S COORDINATES ARE
-8181   12573
1=COM:2=MAN:3=ANOTHER?
COMPUTING COURSE TO
EARTH
DIRECTION= 246.03430746893

DISTANCE= 9302.949468452
OK?Y
WARP?

RETURNED TO EARTH
SETTLE ACCOUNTS
REMAINING GOLD: 8
MARKET PRICE 63 * ON HAND 8 =
504  ———————————— (Depress a key here)
REMAINING CARBON: 30
MARKET PRICE 18 * ON HAND 30 =
540  ———————————— (Depress a key here)

REMAINING ENERGY: 21794 IS
 CONVERTED TO 217
RETURNED EARLY. GET BONUS OF 56
```

```
DIDN'T GO TO SATURN
FINED 2160
EVERYTHING OVER
MONEY= 15106
DAY= 124 PROFIT= 5106
GAME OVER
```

ON n GOTO	Branches to different line depending on the value n.
ATN	Arctangent
SIN	Sine
COS	Cosine
ABS	Absolue value

Chapter 6
Troubleshooting

In creating a substantial program, you will almost always be bothered by bugs. If a program is stopped when an error is generated, refer to the section on error messages of the User's Guide. This chapter gives examples of programs which, although no error is generated, are difficult to execute, and which contain errors that are hard to understand even when you study the error messages. We hope this chapter helps you in debugging programs.

6.1 Endless loop

Problem:

- The same lines are repeated endlessly, and the program does not proceed to the next line.
- Does not accept keys other than the ON/BREAK key.

Explana-: tion

Statements like 10 GOTO 10 is out of question. Endless loop usually occurs when conditional branching is set incorrectly.

■ **The variable type is wrong.**

```
10  DEFINT A — Z
20  PRINT I
30  I = I + 0.1
40  IF I = 1 THEN END
50  GOTO 20
```

Although the programmer apparently tried to increase the value of I by 0.1 each time at Line 30, he/she made a mistake of declaring the variable I as an integer-type variable. Therefore, I always remains 0. The same thing occurs when a FOR ~ NEXT loop is used.

```
10  DEFINT A — Z
20  FOR I = 0 TO 1 STEP .1
30  PRINT I
40  NEXT
```

These programs are written to that the bugs are obvious. However, when actually writing a long program, a programmer might intend to used only intergers, and write a statement like Line 10, and then write a statement like Line 20 when a need for real numbers arises. Therefore, when using variables, always be careful of their type.

■Control variable is changed improperly

```
10  FOR I = 0 TO 10
20  I = I*B
30  PRINT I
40  NEXT
```

This is an example of an usually simple program, but an error like Line 20 does occur when one is writing a program with many statements between a FOR ~ NEXT loop. If the variable B is 0, I remains always 0, and the loop is never terminated.

■Mistake in the error recovery routine

```
10  ON ERROR GOTO 50
20  INPUT A
30  PPINT SQR (A)
40  END
50  A = -A
60  PRINT "IMAGINARY NO.".
70  RESUME
```

This routine is intended not to generate error when the value of A is negative during calculations of square root. However, when this program is executed, it ends up in an endless loop. This is due to the fact that the return from the error recovery routine is specified with RESUME, and therefore, the program tries the same process again and again. Although this program handles negative values with A = −A to calculate their square roots, which is fine, but the PRINT statement in Line 30 is obviously a typo of PRINT. Since this Syntax Error is different from the predicted error (Illegal Function Call due to SQR (−n)), endless loop is generated at Line 30 without any solution. When using an error

recovery routine in a program, do not use the ON ERROR statement until you are sure that there is no unexpected error. It is also a good idea to specify the type of error (check reserved variable ERR or ERL) the error recovery routine will handle.
Also, the declaration of ON ERROR GOTO statement is valid even when the program execution is stopped and put into the direct mode. When an error is generated during the direct mode, the error recovery routine is started (unless the error recovery is stopped with the ON/BREAK key or "Abort" is displayed on the LCD). In such a case, you must execute either ON ERROR GOTO 0, RUN, or NEW command.

6.2 Problems with displays

Problem:

- Numeric values and strings are not displayed properly.

Explanation

Output of characters and numbers is an essential part of a program.

■Problem in handling scrolling.
To display as much information as possible on the LCD, a semi-colon should be placed after each item to be displayed. However, all is displayed continuously (no carriage return/line feed). So, let's think of a program which displays information line by line like the LIST @ command.
Characters to be displayed are read into the variables A$ (0) to A$ (10). When the RETURN key is depressed, the display is advanced.

```
100  FOR I = 0 TO 10
110  PRINT A$ (I);
120  LINE INPUT I$
```

```
130  PRINT
140  NEXT
```

Or , Line 120 can be set to IF INKEY$ = " " THEN 120, or after every 4 lines are displayed, scrolling can be stopped by adding IF MOD 4 = 0 THEN LINE INPUT I$. Furthermore, you should also consider handling execptional cases, such as when A$ (I) is greater than 20 characters (IF LEN (A$ (I) > 19).

■Alignment of numeric values and strings.

Let's say that names within 10 characters are stored in the variables N$ (0) to N$ (10), and telephone numbers within 10 characters are stored in the variables T$ (0) to T$ (10).

```
100  FOR I = 0 TO 10
110  PRINT N$ (I); TAB (10); T$ (I)
120  NEXT
```

The names will be displayed starting from the left corner and telephone numbers starting from the 11th position of the LCD. A word of caution: Even if data consist of only numbers, such as telephone numbers, anything that requires "0" at the beginning should be handled as strings. When they are handled as numeric values, the leading zeroes are omitted as being meaningless. But, there are times when you want to handle numbers as numeric values, and then use them as characters.
Expression for converting a numeric value to a character is as follows:

```
A$ = STR$ (12) ... Content of A$ "12"
A$ = STR$ (– 12) ... Content of A$ = " – 12"
```

As shown , a minus sign precedes a numeric value if it is a negative value, and a blank if it is a positive value. If you know that the numeric values are always positive, or if you do not need a sign, the expression

```
A$ = MID$ (STR$ (X), 2)
```

gives numeric strings without a blank space as in "12".
Also, as in "PRINT 5:5", when numeric values are output in series, you must consider outputting a space after each numeric value, and two spaces between two numbers.
Or, when you want to display number right-justified, use the PRINT USING statement.

Example

```
PRINT USING "# # # #."; 156.21256
      156
PRINT USING "E# #.#*"; 23.567
      E 23.6*
```

For details, refer to "PRINT USING" of the Reference Manual. Also, study how to use the LOCATE and CONSOLE statements.

6.3 Numeric values

Problem:
- Unexpected calculation results.
- Unexpected parameter of instruction occurs, resulting in FC Error.

Explanation: Usually, problem is the name and type of variables.

■An error in variable type

```
10  DEFINT K—Z
20  DEFFNX(A, B)=(A + B)*10000
30  ....
40  ....
```

In a user defined function, such as above, as X is an integer-type variable, FNX is also an integer-type function. Therefore, if the value of A or B is too large, it might cause an OV Error.
Below is an example of an error in converting the type.

```
10  A=10
20  A=A/3
30  A=A*3
40  PRINT G$(A)
```

The programmer intended to output character string G$ (10). However, the value of A is 9.9999999999999 after execution of Lines 20 and 30 according to BASIC. In Line 40, A is used as a subscript (always an integer) of an array, and is truncated to 9. As seen, when using integer-type variables or parameters, it is necessary to keep in mind that the decimals are truncated

■Repetitions and other errors in variable names

```
10  PROG = 10
20  PRICE = 20
30  SUM = PRICE*PROG
```

In this program, the variable PROG and the variable PRICE are treated as the same variable. Since variables are always recognized by the first two characters only, the programmer must be careful in using long names.

■An error in the order of calculation.

```
10  I = 9—5 MOD 3
20  PRINT I
```

Here, the MOD operation which has a calculation priority is calculated before the negative. Accordingly, 5 MOD 3 = 2,9 − 2 = 7 produces the answer, I = 7. It is better to use parentheses such as (9 − 5) MOD 3 or 9 − (5 MOD 3), to avoid these mistakes.

6.4 Other hard-to-recognize errors

Problem: •There seems to be no mistake in the line in which an SN Error has occurred.

Explana-tion: Wrong type of data to be read by the READ statement. In the example below, an SN Error occurs in Line 30

```
10  READ A
20  PRINT A
30  DATA B
```

■ **When an user defined function is used.**

In the next program, SN Error occurs in Line 20.

```
10  DEFFNA (B, C) = 5*(B, C)
20  A = FNA (2, 4)
30  PRINT A
```

Actual SN Error is in Line 10, in which a function is defined.

■ **Hard-to-recognize errors when actually writing a program**

Hard-to-recoginize errors include mixups between "I" and "l", and "O" and "0", and ";" ":", as well as mistakes in the placement of commas. Even more difficult to recognize is two lines that are connected because the RETURN key was not depressed after the first line. Careful scrutiny using the LIST command should reveal these errors.

```
LIST 20
20  INPUT R:A1 = COS (R)
30  BEEP 4, 10
```

Here, lines below Line 20 become part of this line:

```
A1 = COS (R) 30  BEEP 4,10
```

Problem: •The program does not run properly, intervened by RG Error, NF Error or FC Error.

Explanation: An awkwardly written program often causes very hard-to-recognize errors. Several examples follow:

■ON~GOTO statement

```
10  INPUT N
20  IF N>3 THEN 10
30  ON N GOTO 100, 200, 300
40  REM BEEP SUB
50  BEEP H, S:RETURN
100  H=10: S=10: GOSUB 40: GOTO 10
200  H=20: S=20: GOSUB 40: GOTO 10
300  H=30: S=30: GOSUB 40: GOTO 10
```

This program might look strange, but parts like these sometimes occur when writing a long program. The problem lies in the fact that the control goes from Line 30 to Line 40 when N=0. As a result, an RG Error occurs when the control is transferred to RETURN in Line 50. If there is too long a subroutine from Line 40, it is sometimes difficult to find out when the program went into the subroutine. To avoid this, watch out for the possibility of control moving to the next line following an ON ~ GOTO statement, and to always confirm that a subroutine is preceded by either END, STOP, RETURN or GOTO XX.

■Error in which an SN Error is not displayed.

```
10  A$="ABC":FOR I=0 TO 5
20  BEEP I,2:NEXT
30  PRINT A$
```

Here, "ABC: FOR I = 0 TO 5" is assigned as the contents of A$ in Line 10, and the control is transferred to Line 20, resulting in an NF Error. When there are many statements between Lines 10 and 20, an error like the one in Line 10 is often over looked and is hard to find. It is better to avoid unnecessary multi-statements, and use simple statements such as FOR ~ NEXT, GOSUB ~ RETURN, etc. Similar mistakes include that of apostrophe ('). For example:

```
10 PRINT A'THIS IS A SAMPLE
```

will not constitute an SN Error, but is treated as

```
10 PRINT A: REM THIS IS A SAMPLE
```

If this is followed by a RETURN or NEXT statement, the flow of the entire program will become a hopeless jumple.

Canon CANON INC.
7-1, Nishi-shinjuku 2-chome, Shinjuku-ku, Tokyo 160, Japan
P.O. Box 5050, Shinjuku Dai-ichi Seimei Building, Tokyo 160, Japan

CANON U.S.A., INC.
HEAD OFFICE One Canon Plaza, Lake Success, N.Y. 11042, U.S.A.
CHICAGO 140 Industrial Drive, Elmhurst, Illinois 60126, U.S.A.
LOS ANGELES 123 Paularino Avenue East, Costa Mesa, California 92626, U.S.A.
ATLANTA 6380 Peachtree Industrial Blvd., Norcross, Georgia 30071, U.S.A.
DALLAS 2035 Royal Lane, Suite 290, Dallas, Texas 75229, U.S.A.

CANON CANADA INC.
HEAD OFFICE 3245 American Drive, Mississauga, Ontario, L4V 1N4, Canada
CALGARY 2828, 16th Street, N.E. Calgary, Alberta, T2E 7K7, Canada

CANON EUROPA N.V.
P.O. Box 7907, 1008 AC Amsterdam, The Netherlands

CANON FRANCE S.A.
Centre D'Affaires Paris-Nord, Immeuble Ampère 5, 93151 Le Blanc Mesnil, Cedex, France

CANON RECHNER DEUTSCHLAND GmbH.
Fraunhoferstrasse 14, Postfach 8033, München-Martinsried, West Germany

CANON UK LTD.
Waddon House, Stafford Road, Croydon CR9 4DD, England

CANON LATIN AMERICA, INC.
SALES DEPARTMENT P.O. Box 7022, Panama 5, Rep. of Panama
REPAIR SERVICE CENTER P.O. Box 2019, Colon Free Zone, Rep. of Panama

CANON HONG KONG TRADING CO., LTD.
Golden Bear Industrial Centre, 7th Floor, 66-82 Chai Wan Kok Street, Tsuen Wan, New Territories, Hong Kong

CANON AUSTRALIA PTY. LTD.
1 Hall Street, Hawthorn East, Victoria 3123, Australia

Canon

©CANON INC. 1983

PRINTED IN JAPAN

03-149 0783B2.2